April 2014

Indexing Proximity-based Dependencies for Information Retrieval

Samuel Huston
University of Massachusetts - Amherst

INDEXING PROXIMITY-BASED DEPENDENCIES FOR INFORMATION RETRIEVAL

A Dissertation Presented

by

SAMUEL HUSTON

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2014

School of Computer Science

INDEXING PROXIMITY-BASED DEPENDENCIES FOR INFORMATION RETRIEVAL

A Dissertation Presented

by

SAMUEL HUSTON

Approved as to style and content by:

W. Bruce Croft, Chair

James Allan, Member

Andrew McGregor, Member

Weibo Gong, Member

Trevor Strohman, Member

Lori A. Clarke, Chair
School of Computer Science

For Michele Huston

ACKNOWLEDGMENTS

There are a large number of people that I would like to thank for helping me throughout this program. Without their support, the work detailed in this dissertation would not have been possible. First, I'd like to thank my advisor, Bruce Croft, for all his advice and guidance. It is through his advice that I have developed a passion for scientific inquiry. Further, my speedy progress through this program is with thanks to his efforts and guidance. Further, I'd like to thank him for his tireless editing efforts. A direct result of his efforts is that my writing style has improved almost immeasurably during my time in Amherst. I would also like to thank James Allan for all his helpful advice and for all his amusing witticisms. I would particularly like to thank both Bruce and James for their tireless efforts in procuring funding for the lab.

I'd like to thank my mother, Michele Huston. Without her love and support I could not have pursued my interests to this degree. She always encouraged me to find areas that interested me and pursue them, even to the other side of the planet. This thesis is dedicated to her memory. I'd also like to thank my family, both old and new, for all their support throughout this endeavor. Thanks to my new wife Erin, for putting up with the distance between Boston and Amherst for so long, and for helping me in a thousand small ways throughout this degree. Thanks to my father, Geoff, for supporting my choice to move halfway round the globe. Thanks to my sister and brother, Chris and Alice, all for their support. Thanks to my relatively new family-in-law, Kerry, Maureen and Katherine Kravitz, for being so welcoming and supportive.

I'd like to thank Trevor Strohman for many interesting and helpful conversations about the design and operation of large scale information retrieval systems. Trevor was the designer of the Galago retrieval system, upon which much of my research has been built. For all his assistance and advice in my investigations of the utility of sketching techniques for information retrieval problems, I'd like to thank Andrew McGregor. I would like to thank my research collaborators, Alistair Moffat and Shane Culpepper. I have enjoyed many interesting and fruitful discussions with both Alistair and Shane during meetings. Thanks to Vanessa Murdock for many interesting discussions about geo-spacial information retrieval.

I would like to thank the graduate students of the Center for Intelligent Information Retrieval for the many discussions and arguments about the direction of information retrieval research. I've learned a lot through these discussions. Further, I'd like to thank them for listening, having the opportunity to describe research questions helps to refine the questions and ensures that the most informative experiments can be run first. In alphabetical order: Niranjan Balasubramanian, Michael Bendersky, Ethem Can, Marc Cartright, Jeffery Dalton, Van Dang, Laura Dietz, Henry Feild, John Foley, Mostafa Keikha, Weize Kong, Matt Lease, Chia-Jung Lee, Tamsin Maxwell, Don Metzler, Jae Hyun Park, and Zeki Yalniz. I wish them all the best for their future endeavors.

I'd like to thank Dan Parker for his tireless efforts maintaining the cluster of computers that I used to perform many of my experiments. Without his help, many of these experiments would simply not have been possible. I'd like to thank David Fisher for the many pleasurable conversations about my research, and about how to develop Galago. I wish both David and Dan all the best for the future. I'd also like to thank Kate Moruzzi, Jean Joyce and Leeanne Leclerc for helping make sure all of my interactions with the university administration were almost entirely painless.

Finally, I'd like to thank all the friends I've made in the last 5 years for making the university a much more enjoyable place. Thanks to my house mates for helping to make this country feel like home; Elizabeth Russell, Marc Cartright, Dirk Ruiken, Thomas Rossi, David Belanger, Ilene Magipong, Sebastian Magipong, and Orion Magipong. Thanks to all of my friends in the school of computer science for preventing work from feeling too arduous: Aruna Balasubramanian, Emmanuel Cecchet, Jacqueline Feild, Mike Lanighan, Katerina Marazopoulou, Huong Phan, Shiraj Sen, Jennie Steshenko, Laura Sevilla, and Kyle Wray. I'd like to thank some of the friends from the rest of the university partly for providing some perspectives on other fields, and partly for distracting me from the stresses of the program. I'd particularly like to thank Diego Amado and María Turrero for being such good friends throughout my time in Amherst. Finally, I'd like to thank some of my oldest friends for always being able to forgive long periods of silence: Charles Martin, Christina Hopgood, David Edquist, Chris Lim, Lizzy Silver, Kirsty Souter, and Helen Barnes.

This work was supported in part by the Center for Intelligent Information Retrieval, in part by IBM subcontract #4913003298 under DARPA prime contract #HR001-12-C-0015, in part by NSF grant #IIS-0707801, in part by NSF grant #CNS-0934322, in part by NSF grant #IIS-0534383 and in part by NSF CLUE IIS-0844226. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

ABSTRACT

INDEXING PROXIMITY-BASED DEPENDENCIES FOR INFORMATION RETRIEVAL

FEBRUARY 2014

SAMUEL HUSTON

B.C.S. (Hons), UNIVERSITY OF MELBOURNE

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor W. Bruce Croft

Research into term dependencies for information retrieval has demonstrated that dependency retrieval models are able to consistently improve retrieval effectiveness over bag-of-words models. However, the computation of term dependency statistics is a major efficiency bottleneck in the execution of these retrieval models. This thesis investigates the problem of improving the efficiency of dependency retrieval models without compromising the effectiveness benefits of the term dependency features.

Despite the large number of published comparisons between dependency models and bag-of-words approaches, there has been a lack of direct comparisons between alternate dependency models. We provide this comparison and investigate different types of proximity features. Several bi-term and many-term dependency models over a range of TREC collections, for both short (title) and long (description) queries,

are compared to determine the strongest benchmark models. We observe that the weighted sequential dependence model is the most effective model studied. Additionally, we observe that there is some potential in many-term dependencies, but more selective methods are required to exploit these features.

We then investigate two novel index structures to directly index the proximity-based dependencies used in the sequential dependence model and weighted sequential dependence model. The frequent index and the sketch index data structures can both provide efficient access to collection and document level statistics for all indexed term dependencies, while minimizing space costs, relative to a full inverted index of term dependencies. We test whether these structures can improve retrieval efficiency without incurring large space requirements, or degrading retrieval effectiveness significantly. A secondary requirement is that each data structure must be able to be constructed for an input text collection in a scalable and distributed manner.

Based on the observation that the vast majority of term dependencies extracted from queries are relatively frequent in the collection, the "frequent" index of term dependencies omits data for infrequent term dependencies. The sketch index of term dependencies uses techniques from sketch data structures to store probabilistically-bounded estimates of the required statistics. We present analyses of these data structures that include construction and space costs, retrieval efficiency and investigation of any degradation of retrieval effectiveness.

Finally, we investigate the application of these data structures to the execution of the strongest performing dependency models identified. We compare the retrieval efficiency of each of these structures across two query processing algorithms, and across both short and long queries, using two large web collections. We observe that these newly proposed data structures allow the execution of queries considerably faster than when using positional indexes, and as fast as a full index of term dependencies, but with lowered storage overhead.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Term Dependencies in Information Retrieval Models

The bag-of-words view of documents has been remarkably effective in the field of information retrieval. This view asserts that all terms are independent of the other terms in the document, such that any document, $\mathcal{D}$, and any random permutation of the terms in this document, $\mathcal{D}'$, are considered identical. A simple probabilistic view of term independence is expressed as:

$$P(t_1, t_2) = P(t_1)P(t_2)$$

Cooper (1991) presents a good summary of the different types of assumptions of term independence and how they should, and should not, be combined. Assumptions of term independence have also been formalized in other information retrieval model frameworks. The vector space retrieval model, introduced by Salton et al. (1975), asserts that each unique term in the vocabulary of a collection is represented by its own dimension in a high-dimensional space.

Generally, assumptions of term independence have been made to enable scalable, efficient and effective information retrieval models. This is evidenced by the prevalence of bag-of-words models in open source search engines. Indri[1] and Galago[2] both use the query likelihood retrieval model with Dirichlet smoothing by default (Song

[1] A component of The Lemur Project, http://www.lemurproject.org/indri.php

[2] A component of The Lemur Project, http://www.lemurproject.org/galago.php

and Croft, 1999). Lucene[3] uses the combination of a Boolean retrieval model for document selection and a vector space model for document ranking (Salton et al., 1975). Terrier[4] defaults to the *PL2* model that is based on the Divergence From Randomness (DFR) retrieval model framework (Amati and Van Rijsbergen, 2002). Finally, while it is not the default retrieval model in these open source search engines, the *BM25* retrieval model is commonly used as a baseline in information retrieval research (Robertson and Walker, 1994). All of these retrieval models assume term independence.

However, the assumption that the order of terms and the relationships between them carry no information is intuitively flawed. For example, each of the following sequences of text are considered identical under the assumption of term independence:

```
"you do not have to be mad to be here but it helps"
"you do have to be here not to be mad but it helps"
"to be do here not but you helps to be mad have it"
```

This observation has lead to many attempts to incorporate term dependencies into information retrieval. Recent attempts have shown consistent improvements over bag-of-words models. Building on the Language Model retrieval framework, the Markov Random Field models (MRF) (Metzler and Croft, 2005) uses two types of term dependencies: ordered and unordered windows. Bendersky et al. (2011) extend the MRF by introducing additional weighting parameters. Weighted Concept Models (Bendersky and Croft, 2008, Bendersky et al., 2010) use a set of key-concept phrases extracted from queries to improve retrieval performance. The key-concept phrases are based on noun phrases extracted from linguistic analysis of the query. Park et al. (2011) present the Quasi-Synchronous Dependence Model that improves retrieval performance through statistics gathered for a set of syntactic relations between pairs of

[3]http://lucene.apache.org/

[4]http://terrier.org/

terms. Maxwell and Croft (2013) demonstrate strong retrieval performance using dependency parsing techniques to extract dependencies.

Along a different line of research, the BM25 model (Robertson and Walker, 1994) [5], a bag-of-words retrieval model, has been extended to use term dependency statistics in several different methods. Rasolofo and Savoy (2003) introduce scores based on a term proximity function into the BM25 function. Svore et al. (2010) propose a bigram operator that is similar to this term proximity function. Song et al. (2008) use spans of text to compute the distance between queried terms in each document. Each of these term dependency extensions to the BM25 model has been shown to improve retrieval performance over the baseline model.

Positional Language Models (Lv and Zhai, 2009) offer a different method of defining term proximity functions. A smoothed language model is defined for each location in a document, such that each term influences neighboring locations at a decayed rate. In this manner, the language model, defined at each location of the document, directly encodes the proximity of all query terms in the document. Documents are then retrieved based on an aggregate value across locations.

This is not a comprehensive list of retrieval models or term dependency features. Many other retrieval models exploit relationships between terms to improve retrieval performance. We will discuss the relationship between these retrieval models and the contributions of this thesis in Chapter 2.

We define a term dependency to be any relationship between 2 or more terms in the context of a text document. This definition includes natural language structures, such as noun phrases, verb phrases, term relationships extracted from parse trees and part-of-speech annotations. It also includes term proximity functions, such as n-grams,

[5] While this paper is generally cited for the BM25 retrieval model, the BM25 scoring function is not actually defined in this paper. This retrieval model is defined in proceedings of TREC-3, in the paper "Okapi at TREC-3" (Robertson et al., 1992).

ordered windows, unordered windows, or spans of text. Dependency retrieval models are defined to be any retrieval model that exploits term dependencies for information retrieval. In this dissertation, we will focus on proximity-based dependency models.

Current research almost exclusively uses bag-of-words models as state-of-the-art benchmarks. For example in the 34^{th} ACM SIGIR conference (2011), six new retrieval models were presented. We note that all six compare results with at least one bag-of-words model, but only one compares results with a dependency model. The infrequent use of dependency models as strong baselines is a problem for the field. It has even led to some researchers questioning whether ad-hoc retrieval has improved significantly in the last decade (Armstrong et al., 2009a).

The difference in efficiency between bag-of-words and dependency models is one of the causes of this problem. Web search engines, and many other production information retrieval systems, are required to return results to users in subsecond times. Computing these dependency models at scale, using current technology is not feasible without taking rank-unsafe short cuts, or requiring huge amounts of storage for precomputed results.

The aim of this thesis is to investigate methods of improving the efficiency of term dependency retrieval models, and thereby support further research into the use of term dependencies in information retrieval. However, to determine the most effective dependency models, we start by comparing a variety of proximity-based dependence models. Based on the results of this comparison, particular focus will be made on improving the sequential and full dependence retrieval models proposed by Metzler and Croft (2005), and the weighted sequential dependence model proposed by Bendersky et al. (2011). We describe both of these retrieval models in detail in Chapter 4.

These dependence models use two types of term dependency features: ordered and unordered windows. These features are directly used by several other dependency

retrieval models (Bendersky and Croft, 2008, Bendersky et al., 2010, Peng et al., 2007, Rasolofo and Savoy, 2003, Xue and Croft, 2011). They are also a good surrogate for linguistic features such as phrases and dependency tree relationships. We assert that the optimizations for the sequential and full dependence models will be applicable to many other term dependency models.

1.2 Data Structures for Dependence Models

In order to execute dependency models, as opposed to bag-of-words models, additional data must be stored in the index. We define an index here as a set of data structures designed to store and return the collection and document statistics required to compute a retrieval model function for a given query, over each document in a collection.

The inverted index structure has been developed as an efficient method of storing statistics for term occurrences within a collection of documents. This structure can be considered to be a mapping from terms or term identifiers to posting list data. A posting list is defined as an ordered sequence of document-level statistics for the term or term identifier. Inverted indexes are commonly too large to be stored in memory. For this reason, they are sometimes referred to as file organizations. A good introduction to this data structure and some variations is presented by Witten et al. (1999).

This definition allows for wide variation in implementation, including memory and disk-based implementations, the use of hash-based or tree-based mapping structures, and many different compression schemes. Generally, implementations focus on minimizing space usage without compromising the efficient extraction of posting list data.

In order to execute a query using a specific retrieval model, the statistics stored in the posting lists associated with the queried terms are extracted. If the index is

	cf_t, dc_t	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$	$\mathcal{D}_5$	$\cdots$
$term_0$	1123, 530	1	0	2	6	0	$\cdots$
$term_1$	33, 22	2	0	0	1	2	$\cdots$
$term_2$	101, 32	0	5	1	0	0	$\cdots$
$\vdots$							

Figure 1.1: Example matrix representation of a term-level, non-sparse inverted index. The first two integers in each posting list are the collection frequency, and the document count of the term. Then, each cell stores the frequency, $f_{t_j,\mathcal{D}_i}$, of the term, t_j, in document $\mathcal{D}_i$.

stored on disk, buffered stream readers can be used to efficiently read the posting list data, ensuring a minimal number of random disk accesses are performed.

Conceptually, an inverted index can be viewed as a matrix, where each cell in the matrix stores the frequency of a term in a particular document. Figure 1.1 shows an example matrix inverted index. As can be seen in this example, the matrix representation is commonly very sparse, with most documents containing only a tiny fraction of the vocabulary of the collection of documents.

Given this observation, it is natural to consider sparse representations that only store non-zero entries. One example is the inverted count index. In the literature, this data structure has been given many different names, including 'frequency index'. In this dissertation, we use the name 'inverted count index' or 'count index'. This nomenclature is intended to avoid confusion with the 'frequent index' that stores only frequent terms and term dependencies, to be defined in Chapter 6.

An example of the inverted count index is shown in Figure 1.2. To support efficient access to collection statistics, the collection frequency and document count are stored at the start of each posting list. This structure allows the execution of a wide variety of bag-of-words retrieval models in a space and time-efficient manner. It does not support the execution of dependency models because location data is not stored.

$term_0$	1123, 530	$\langle \mathcal{D}_1, 1 \rangle$, $\langle \mathcal{D}_3, 2 \rangle$, $\langle \mathcal{D}_4, 6 \rangle$, $\cdots$
$term_1$	33, 22	$\langle \mathcal{D}_1, 2 \rangle$, $\langle \mathcal{D}_4, 1 \rangle$, $\langle \mathcal{D}_5, 2 \rangle$, $\cdots$
$term_2$	101, 32	$\langle \mathcal{D}_2, 5 \rangle$, $\langle \mathcal{D}_3, 1 \rangle$, $\cdots$
$\vdots$		

Figure 1.2: Example inverted count index, a sparse representation of the document-frequency matrix. The first two integers in each posting list store the collection frequency and the document count of each term, the posting list stores a list of document identifiers, and the document frequency of the term. Documents omitted from the posting list are asserted to have a zero document frequency.

$term_0, term_1$	113, 32	$\langle \mathcal{D}_3, 3 \rangle$, $\langle \mathcal{D}_4, 5 \rangle$, $\langle \mathcal{D}_7, 2 \rangle$, $\cdots$
$term_1, term_3$	40, 25	$\langle \mathcal{D}_2, 3 \rangle$, $\langle \mathcal{D}_5, 2 \rangle$, $\langle \mathcal{D}_9, 1 \rangle$, $\cdots$
$term_1, term_5$	73, 46	$\langle \mathcal{D}_1, 5 \rangle$, $\langle \mathcal{D}_8, 1 \rangle$, $\cdots$
$\vdots$		

Figure 1.3: Example inverted count index of 2-grams. This structure stores a posting list for each 2-gram extracted from a collection of documents.

In dependency retrieval models, collection- and document-level statistics for term dependency features, such as n-grams, are required, in addition to similar statistics for terms. A simple method of obtaining these statistics is to directly index the desired term dependencies. An example inverted count index of 2-grams is shown in Figure 1.3. We refer to this structure as a "full" index, to distinguish it from an inverted count index of terms.

An important variation on the inverted count index is the inverted positional index (Witten et al., 1999). In addition to the data stored in the inverted count index, the inverted positional index structure, or 'positional index', stores the offsets of each term in each document. Figure 1.4 shows an example inverted positional index of terms. Positional data allows the reconstruction of a wide variety of term dependency features at query time. Note that this structure contains all of the data

$term_0$	1123, 530	$\langle \mathcal{D}_1, 1, [4] \rangle$, $\langle \mathcal{D}_3, 2, [14, 25] \rangle$, $\langle \mathcal{D}_4, 5, [1, 5, 26, 50, 100] \rangle$, $\cdots$
$term_1$	33, 22	$\langle \mathcal{D}_1, 2[14, 25] \rangle$, $\langle \mathcal{D}_4, 1, [15] \rangle$, $\langle \mathcal{D}_5, 2, [4, 19] \rangle$, $\cdots$
$term_2$	101, 32	$\langle \mathcal{D}_2, 5, [50, 53, 60, 74, 99] \rangle$, $\langle \mathcal{D}_3, 1, [62] \rangle$, $\cdots$
$\vdots$		

Figure 1.4: Example inverted positional index, a sparse representation of the document-frequency matrix. The first two integers in each posting list store the collection frequency and the document count of each term, the posting list stores a document identifier, the document frequency of the term, and a list of locations at which the term occurs in the document. Documents omitted from the posting list are asserted to have a zero document frequency.

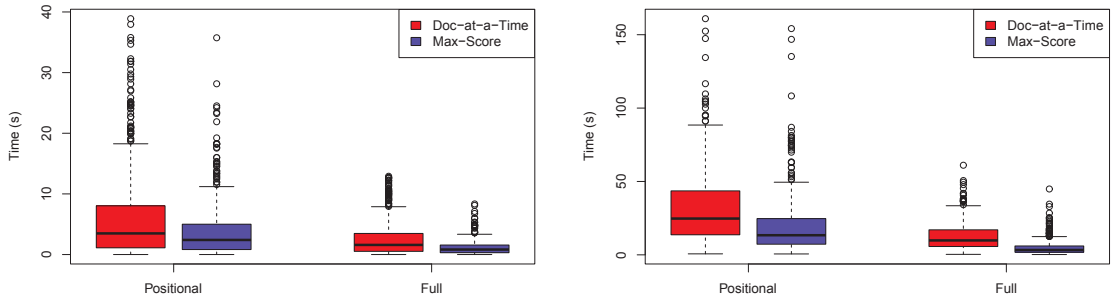
stored in the inverted count index, and can be used to construct any entry in the full index.

Alternative index structures, implementations, and compression schemes will be discussed in Chapter 2. Chapter 5 will further discuss and investigate the benefits and costs of these index data structures for dependency features. We investigate the utility of the frequent index structure for term dependencies in Chapter 6. We propose and investigate the sketch index structure in Chapter 7.

1.3 Efficiency of Dependency Models

Existing technology to evaluate dependency retrieval models over large textual collections generally requires an unreasonable amount of processing time or makes unreasonable space requirements. The positional index is able to execute a wide range of dependency models, but costly comparisons of positional data is required at query time. The full index eliminates the need for costly comparison operations at query time, but it makes very large space requirements.

We use the Document-at-a-Time (shown in red) and Max-Score (shown in blue) algorithms to execute 500 short queries and 500 long queries sampled from three TREC Million Query Tracks (2007, 2008, and 2009). In this experiment, a



(a) Short queries

(b) Long queries

Figure 1.5: Query processing times for 500 short and 500 long queries sampled from the TREC Million Query Track, for the sequential dependence model. Queries are processed using two types of indexes; full indexes (Full) and positional indexes (Positional), and two types of query processing algorithms, the Document-at-a-Time algorithm, and the Max-Score algorithm. Each set of queries is executed 5 times with randomized query execution order.

short query is defined as 2 or 3 terms, and a long query is defined as 4 to 12 terms. Both query processing algorithms are defined and discussed in Chapter 8.

Figure 1.5 shows the distribution of query processing times for the sequential dependence model (Metzler and Croft, 2005) using a positional index and a full index, of the TREC GOV2 collection. The distribution of query processing times for each experimental setting is shown using a box-and-whisker plots. Each colored box spans the 1^{st} to 3^{rd} interquartile range of the data. The whiskers attached to each box indicate the 95% confidence interval of the interquartile range. The bar across each box indicates the median query processing time. Outliers, which fall outside of the 95% confidence interval, are indicated with circles.

It is clear from the plotted execution times that the sequential dependence model is much slower to execute using positional indexes, than using full indexes. Averaging across all timed executions of both query processing algorithms, and both types of queries, we observe that full indexes offer a 65% reduction in query processing time

Table 1.1: Space requirements of postional and full index structures for the TREC GOV2 collection.

Index	Vocab. (GB)	Postings (GB)	Combined (GB)
Positional	0.41	42.9	43.3
Full	33.5	261	294

over positional indexes. Unsurprisingly, we observe a large difference in query execution time between short and long queries. This is due to a rise in the number of query features that require statistics to be extracted from the index. Other comparisons are possible from this data. We further extend this experiment and discuss results in more detail in Chapter 8.

This timing data suggests that we should avoid the use of positional indexes, in favor of full indexes. However, we must also consider the space requirements of the full index structure. Table 1.1 shows the space requirements of both types of indexes, for the GOV2 collection. We can see that the space requirements of the full index are almost 7 times that of the positional index.

This investigation into trade-offs between space requirements and retrieval efficiency for these existing data structures, and for the new data structures proposed in this thesis, is expanded in Chapter 8.

1.4 Contributions

In this thesis, we perform an extensive comparison of proximity-based dependency models. Based on the results of this comparison, we develop index structures and indexing strategies to improve the retrieval efficiency of the strongest performing dependency models, without incurring unreasonable time or space costs, and without degrading retrieval effectiveness. The performance of these structures is tested empirically on a variety of collections. Finally the application of these index structures

to the execution of these dependency models is investigated. We now detail each of the major contributions of this dissertation.

1 **The first extensive comparison of existing proximity-based dependency models**

We compare a wide range of bi-term dependency models across a range of collections and both long and short queries. We then use the strongest performing bi-term retrieval models to investigate into the relative utility of many-term dependency features, as compared to bi-term dependency features.

2 **The strongest performing proximity-based dependency model is the internal variant of the weighted sequential dependence model**

We observe that the internal variant of the weighted sequential dependence model consistently improves retrieval performance over each of the other bi-term and many-term dependence models. The Fisher randomization test shows that these performance improvements are statistically significant in many settings. This result shows that the weighted sequential dependence model is an appropriate benchmark for future investigations of retrieval features.

3 **There is no empirical difference between three very different algorithms for the extraction of ordered and unordered windows**

The extraction of ordered and unordered windows from positional data depends on assumptions of term reuse. We present three possible assumptions, and three corresponding window extraction algorithms. We observe that there is a small difference between the collection frequencies of windows as extracted by each of these algorithms. Further, we observe that there is no difference in retrieval effectiveness between the algorithms, and similarly almost no difference in retrieval efficiency.

4 The space requirements of the full index can be estimated directly from information detailing the skew of the particular type of ordered or unordered window

We investigate the distribution and vocabulary skew of proximity-based term dependencies. We show that this data can be used to accurately estimate the space requirements for full index structures of various different ordered and unordered windows. Further, we observe that there are well defined patterns that allow estimation of full indexes of a variety of window-based term dependencies.

5 Analysis of the frequent index for information retrieval

We analyze the use of a frequent index for information retrieval. This contribution includes analysis of novel, efficient indexing algorithms for monolithic and parallel systems, analysis of retrieval efficiency, and investigation into the impact on retrieval effectiveness when used in a lossy manner. We observe that this type of index can reduce space requirements relative to a full index, without compromising retrieval performance.

6 Analysis of the sketch index for information retrieval

We define the sketch index structure for storing term dependency statistics. We present a discussion of the probabilistic error rate for estimated term dependency statistics. We analyze the impact on retrieval efficiency of the use of the sketch index structure, and an investigation into its impact on retrieval effectiveness.

7 Comparison of each of these term dependency indexes for the execution of the strongest performing dependency retrieval models

Finally, we empirically evaluate the impact each term dependency index structures has on the execution of the most effective proximity-based dependency models for two efficient query processing algorithms.

1.5 Outline

The structure of this dissertation is as follows.

- Chapter 2 presents related work, and discusses where and how the research conducted in this thesis is applicable to previous research.
- Chapter 3 describes and presents statistics for the collections and query sets that will be used throughout this dissertation. It also presents each of the retrieval metrics and statistics tests used in the thesis.
- Chapter 4 presents an extensive comparison of state-of-the-art proximity-based dependency models for ad-hoc retrieval. It also details an investigation into the utility of many-term dependency features, in comparison to bi-term dependency features.
- Chapter 5 describes current approaches to storing and retrieving term dependency statistics. It investigates the properties of several types of term dependencies for a set of English corpora. It also analyses the efficiency of extracting proximity-based dependency statistics from the positional index data structure.
- Chapter 6 presents the frequent index data structure for term dependency data. It presents experiments testing the scalability of various different index construction algorithms. It also presents an investigation into the effect frequency thresholding has on retrieval effectiveness.
- Chapter 7 presents the sketch index data structure, and discusses the theoretical guarantees of the structure. It also presents empirical experiments investigating the retrieval efficiency, retrieval effectiveness and the observed error in statistical estimations over large English corpora.
- Chapter 8 investigates the integration of term dependency data structures into existing query processing models, in order to execute the most effective retrieval

models. Experiments in this chapter focus on trade-offs between retrieval efficiency, and space requirements for each of the term dependency indexes.

- Chapter 9 summarizes the contributions made in this body of work and discusses potential future directions for more research in this area.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this Chapter we discuss background information and previous research relating to the contributions made in this thesis. We start by discussing previously proposed methods of extracting dependencies from queries, and different methods of modeling these extracted term dependencies for information retrieval. We then discuss proposed models for the related problems of discovering or searching for instances of partial document duplication and local test reuse. We discuss related, alternative index structures for information retrieval, including compression techniques. We then discuss previous research on query processing algorithms that extract the data stored in the index structures and execute a specific retrieval model, in order to rank each document in the collection. Finally, we discuss various different caching structures that have been proposed to improve the efficiency of information retrieval systems in production environments.

2.1 Modeling Dependencies

Some early approaches to the use of dependencies between terms in information retrieval includes work by Wong et al. (1985), van Rijsbergen (1977), Losee (1994) and Yu et al. (1983). A problem for these early approaches to term dependency features is that each of these models introduces significant complexity into proposed retrieval models, without showing consistent improvement over bag-of-words retrieval models. In a survey of the literature, Salton and Buckley (1988) state:

> "... the judicious use of single term identifiers is preferable to the incorporation of more complex entities"

However, more recent publications have started to demonstrate consistent improvements through the modeling of dependencies (Metzler and Croft, 2005, Bendersky et al., 2010, Xue and Croft, 2012). A key contribution of this thesis is an extensive comparison of proximity-based dependency models to determine the relative value of many-term dependency models, when compared to bi-term dependency models, and bag-of-words models.

The use of dependencies in information retrieval can be split into two related problems. First, term dependencies, or groups of dependent terms, must be detected in, or generated from, the input query. Second, each term dependency must be matched to instances in the document collection, and the score or probability of relevance or score of the document is modified according to statistics extracted from these matches. This description allows for the detection and matching of many different types of dependency features.

2.1.1 Extracting Dependencies from Queries

The simplest method of identifying groups of dependent terms in a query is to assume that all query terms depend on all other query terms. Several dependency models make this assumption. BM25-TP (Rasolofo and Savoy, 2003) extracts all pairs of terms from the query. Similarly, the full dependence model (FDM) (Metzler and Croft, 2005) uses each group in the the power set of query terms. A key problem for both of these models is that the number of extracted dependencies grows exponentially with the number of query terms, making longer queries impractical.

Two retrieval models, BM25-Span (Song et al., 2008) and the positional language model (PLM) (Lv and Zhai, 2009) also make the assumption that all query terms

depend upon all other query terms. However, both models only produce a single group of dependent terms, the group of all query terms.

To improve efficiency, a common alternative is to assume that only adjacent pairs of query terms are dependent. The n-gram language models approach presented by Song and Croft (1999) is an example of this method. Indeed, this approach has been used effectively by many successful dependency models (Bendersky et al., 2012, 2010, Metzler and Croft, 2005, Peng et al., 2007, Svore et al., 2010, Tao and Zhai, 2007). A key advantage of this dependency assumption is that even long queries remain computationally feasible after the inclusion of all dependency features.

While, in general, pairs of adjacent terms capture useful information in the query, the assumption of positional dependence also has the potential to introduce misleading pairs of terms. For example, consider extracting all sequential term pairs from the query: *desert fox news stories*. While the query intent may have been to find contemporary articles about the WWII field marshal, Erwin Rommel, the assumption of sequential dependence leads to the extraction of *"fox news"*. This pair of terms has the potential to mislead a retrieval model to focus on the news channel, rather than the desired information. Even so, retrieval models that use the sequential dependence assumption have been shown to improve average retrieval performance compared to bag-of-words retrieval models.

Several different linguistic techniques have been used in an attempt to improve dependency extraction. Srikanth and Srihari (2003) use syntactic parsing to identify concepts in queries. Gao et al. (2004) propose a method of extracting dependencies using a linkage grammar. Park et al. (2011) use quasi-synchronous parsing to generate syntactic parse trees, from which dependencies are extracted. Maxwell and Croft (2013) have recently shown retrieval performance can be improved through the use of dependency parsing techniques in the extraction of non-adjacent subsets of dependent terms. A common requirement of these methods is a natural language query.

Query segmentation has also been proposed as a method of extracting only the most informative dependencies from an input query (Bendersky and Croft, 2009, Bergsma and Wang, 2007, Jones et al., 2006, Risvik et al., 2003). Typically, these methods use a set of features to determine where to segment (or separate) the query. Detected segments are then considered term dependencies. Bendersky and Croft (2008) extend this work to classify a subset of detected query segments as key concepts.

None of these techniques have been shown to be consistently better than the simpler adjacent pairs method for identifying good candidate term dependencies. In addition, they commonly rely on external data sources such as linguistic models, Wikipedia articles, and query logs. For this reason, as stated in the introduction, we decided to simplify much of the work in this thesis by leaving the investigation of the non proximity-based methods to future work. However, we assert that the dependency model comparison performed in Chapter 4 can be used to determine an appropriate benchmark for a future studies.

2.1.2 Dependence Retrieval Models

Given that a group of terms has been identified as dependent, a dependency model must also specify how to match and score each group of dependent terms for each scored document. Matching methods commonly focus on measuring how often the dependent group of terms occurs in the document, subject to a proximity constraint. Scoring methods are generally based on the term scoring techniques used in bag-of-words models. Note that several of the dependency models discussed in this section will be investigated in more detail in Chapter 4. In that chapter, we compare proximity-based bi-term and many-term dependency models that do not require external data sources, such as query logs, Wikipedia, or linguistic models. In this section, we identify each model that fits these restrictions as it is presented, and

we present a more detailed discussion of each of the compared dependency models in that Chapter 4.

One of the simplest approaches is to match dependencies as n-grams or phrases. This method reports a match when all terms in a dependent group occur sequentially in a document. This type of feature has been used in several dependency models (Bendersky et al., 2010, Metzler and Croft, 2005, Song and Croft, 1999, Srikanth and Srihari, 2003). From a scoring and weighting perspective, an occurrence of a dependent group is generally treated similarly to the occurrence of a single term.

Another common method of matching a dependent group to instances in a document is to count text windows, or constrained regions of the document, that contain all the dependent terms. Window-based features have been used in a number of retrieval models (Metzler and Croft, 2005, Peng et al., 2007, Rasolofo and Savoy, 2003, Tao and Zhai, 2007). Similar to the phrase-type features, window-based occurrences may be evaluated as terms (Bendersky et al., 2010, Metzler and Croft, 2005, Peng et al., 2007). Alternative approaches have also been proposed. For example, Rasolofo and Savoy (2003) propose a method that scores windows of term pairs proportional to the inverse distance between each matched instance of the pair of terms.

Matching each detected term dependency twice, as both a phrase and as a window feature has been shown to result in a very effective retrieval model. The sequential dependence model (SDM), proposed by Metzler and Croft (2005), has become a benchmark against which new retrieval models are tested. For example, Lease (2009) and Bendersky et al. (2010) apply learning methods to improve the weighting of the features used in SDM. Xue et al. (2010) present a method of reducing verbose queries to a subset of the most important terms. They then construct several different retrieval models using these query subsets in conjunction with the query likelihood and sequential dependency models. The best performing of these retrieval models each incorporates the sequential dependency retrieval model as a weighted component. In

Chapter 4, we include both SDM, and a variant of the weighted sequential dependence model (WSDM) (Bendersky et al., 2010) in the comparison of dependency models.

Xue and Croft (2010) present a method of combining several different retrieval models. This model applies a set of different query reformulations, and scores documents as the weighted geometric mean of the scores produced by each reformulation. Variants of this model have been shown to improve retrieval effectiveness over several strong baselines (Xue and Croft, 2011, 2012). The final structured queries produced by these variants each incorporate several instances of the sequential dependency model. In particular, reformulation trees (Xue and Croft, 2012) use the sequential dependency model to produce a document probabilities for each leaf node in the tree.

Peng et al. (2007) discuss methods of incorporating windows into the divergence from randomness framework (DFR). The score of a document is defined to be a linear combination of the DFR-PL2 model (Amati and Van Rijsbergen, 2002) and a newly defined proximity function based on the binomial randomness model (Macdonald et al., 2005). However, authors assert that any DFR model would be suitable for scoring window features. Two proximity-based DFR models are included in the comparison of dependency models presented in Chapter 4.

Several attempts have been made to introduce term dependency features into the BM25 retrieval model. Rasolofo and Savoy (2003) introduce a bi-term proximity function, where windows of width five or less contribute to the aggregate score. Büttcher et al. (2006) propose a similar bi-term proximity function. A key difference is that their model permits arbitrary width windows to be extracted from each scored document. Svore et al. (2010) propose a frequency based scoring function to replace the distance based function in BM25-TP. We include BM25-TP in our comparison of dependency models in Chapter 4.

Mishne and de Rijke (2005) investigate the introduction of phrase and proximity operators into the vector-space model (Salton et al., 1975). They integrate term

dependencies by redefining a term in the vector space model as either a term, a phrase term or a proximity term. All sequential n-grams extracted from an input query are then matched to instances in each document as either phrases or proximity terms. Ordered and unordered windows used in SDM (Metzler and Croft, 2005) are very similar to the phrases and proximity terms investigated by Mishne and de Rijke (2005).

We assert that the term dependency index structures analyzed in this thesis, in Chapters 5, 6 and 7, can each be directly applied to improving the efficiency of executing all of these retrieval models. We also note that all of these models make use of several features extracted from the collection and some external data sources in determining the weight of each feature or the set of reformulated queries to use. Several of these features are themselves modeled as ordered and unordered windows. Therefore, the contributions made in this thesis can be used to improve the efficiency of the generation and weighting of queries using these retrieval models. However, we leave the quantitative evaluation of any efficiency improvements for BM25, DFR, and vector space dependency models to future work.

Several methods of matching term dependencies to instances in documents using linguistic features have also been explored in previous work Park et al. (2011), Gao et al. (2004), Nallapati and Allan (2002). Each of these retrieval models require that documents are parsed into similar relationships between terms. Assuming that a collection of documents can be parsed to extract linguistic dependencies present in each document, we assert that term relationships can be extracted and indexed using the index structures studied in this thesis. We leave this investigation to future work.

Another approach is to cluster all dependent term occurrences into text spans (Song et al., 2008, Svore et al., 2010). Each span is evaluated according to the number of matching terms in the span and size of the span, where spans of a single term use

the maximum span size. For these models, the definition of a span is implied by the specific algorithm proposed to cluster term occurrences into spans.

Positional language models (PLM) (Lv and Zhai, 2009) propose a method of modeling dependencies between all query terms, without matching specific dependency instances. Dependencies are implicitly modeled through the evaluation of a specific document position. Each matching query term instance in the document propagates a partial count to the position to be evaluated, through a kernel function. The score, or value, of any specific position is proportional to the proximity of each query term to the position.

Span-based models and PLM extract just a single term dependency from the query, consisting of all query terms. The index structures studied in this thesis cannot be directly applied to this type of term dependency. Any index structure for this type of dependency would need to map entire queries to a posting list of scores for documents. Clearly, this index closely resembles a cache structure that stores recently the top k documents for a set of queries. We include the BM25-Span model, and two PLM models in our comparison of dependency models, in Chapter 4.

Tao and Zhai (2007) propose several different aggregate distance functions over the set of all matched query terms in the document. In their study, documents are scored using KL divergence or BM25 combined with a retrieval score "adjustment factor". The adjustment factor transforms an aggregate measure of distance, over all matched term instances in the document into a "reasonable" score contribution. The aggregate functions tested include the minimum, maximum, average, span and min-cover distances. They observe that the minimum distance function in this model optimizes performance. However, the authors also observe that this best performing model is not statistically different from the sequential dependence model. For this reason, we omit the evaluation of these models from this work.

The optimal size of term dependency is investigated by Bai et al. (2008). They use a proximity operator that operates on windows of size 16. Each window is scored according to the number of terms missing and the distance between the present terms for the term dependency extracted from the query. They show that there is some value in longer n-grams, with 3- to 5-term dependencies optimizing their evaluation criteria. The inclusion of the missing terms in the proximity function makes this particular type of term dependency operator infeasible to index.

Some learning-to-rank retrieval models also use dependency features. For example, Svore et al. (2010) investigate the effectiveness of several BM25-based dependency models, that are decomposed into features in a learning-to-rank retrieval framework. Their study also introduces new features that are detected in matched spans. Using a large set of training data, (27,959 queries), they are able to show large retrieval effectiveness improvements over the original formulations. A combination of a lack of training data, and the large number of parameters in these proposed learning-to-rank retrieval models, make evaluation of these models infeasible for this study.

2.2 Partial and Local Duplicate Detection

So far, we have focused on the use of term dependencies for ad-hoc search. Each of the index structures presented in this thesis can also be used to improve the efficiency of a partial duplicate detection system. There are two related tasks in this area; search and detection. In the search task, all documents that duplicate some portion of a given document must be identified and returned. In the detection task, all document duplicates in a collection must be identified and returned. We focus on the former task in this section. The most difficult aspect of this task is the ability to detect relatively small amounts of duplication, or local text reuse.

Many previous approaches to duplicate detection rely on larger n-grams or shingles (Manber, 1994, Schleimer et al., 2003, Brin et al., 1995). In general, these al-

gorithms operate by constructing a reduced representation, or fingerprint, of each document. Commonly a fingerprint consist of a set of n-grams, shingles or phrases extracted from the document.

A variety of algorithms have been proposed that create document fingerprints by selecting a subset of all n-grams in each document. A key requirement of this algorithm is determinism, if an n-gram is selected in one document, it must also be selected from all other documents. Manber (1994) present the $0 \mod p$ algorithm. This algorithm constructs a fingerprint using all n-grams that hash to 0, where the hash value is restricted to the domain p using a modulus function. Winnowing (Schleimer et al., 2003) is a similar algorithm, however, it guarantees a maximum distance between selected n-grams.

Brin et al. (1995) present non-overlapping approach to constructing a fingerprint. The hash breaking algorithm document terms are grouped. Each term that is selected using the $0 \mod p$ function, indicates the end of the current phrase. Seo and Croft (2008) present a method of further reducing the space requirements of each fingerprint, by using a Discrete Cosine Transform (DCT) function.

We assert that both the frequent index and the sketch index presented in this thesis can be used to efficiently store and retrieve each fingerprint component. Where the space requirements to store larger n-grams can be sufficiently low that it may be feasible to completely avoid the sampling methods used above. However, investigation and analysis of the application of these index structures to the problem of local text reuse detection is beyond the scope of this thesis.

2.3 Document-ordered Inverted Index

In this thesis, we focus on the construction and use of document-ordered inverted index structures (Witten et al., 1999). To reduce space requirements, each document is assigned an identifier. For each indexed term, the data structure stores a series of

postings, one for each document. This sequence of postings is referred to as a posting list. The term "document-ordered" refers to the requirement that the posting list is stored in increasing order of document identifier. Generally, retrieval algorithms open several posting lists, and process them in parallel, by iterating through the set of document identifiers present in one or more posting lists.

By storing documents in increasing order, differential encoding, sometimes called d-gap encoding, can be used to reduce the space required to store document identifiers. As an example, the following posting list shows the set of document identifiers corresponding to documents that contain the term 'term', and the frequency of the term in the document $\langle d, f_{t,d} \rangle$:

$$\text{term} \ : \langle 1, 2 \rangle, \langle 3, 1 \rangle, \langle 4, 1 \rangle, \langle 5, 2 \rangle, \langle 7, 5 \rangle, \langle 9, 1 \rangle, \langle 10, 1 \rangle$$

Applying differential encoding, this posting list is represented:

$$\text{term} \ : \langle 1, 2 \rangle, \langle +2, 1 \rangle, \langle +1, 1 \rangle, \langle +1, 2 \rangle, \langle +2, 5 \rangle, \langle +2, 1 \rangle, \langle +1, 1 \rangle$$

Document identifiers are decoded by maintaining a running total of all previous d-gap values stored in the posting list.

This transformation changes the distribution of the set of integers stored in each posting list. Common terms, with long posting lists, will tend to contain a large number of very small values, and rarer terms, with shorter posting lists, will contain relatively high values. So, the most common integers stored will be low, allowing effective use of variable width numeric compression schemes.

A downside to this type of encoding is that skipping over a number postings to score a specific document requires the decoding of all intermediate document identifiers. Skip lists allow skipping of blocks of data by storing the document identifier for

the first document in each block. A pointer to the start of the next block is stored at the head of each block.

$$\text{term} \ : [11, \langle \mathbf{1}, 2\rangle, \langle +2, 1\rangle, \langle +1, 1\rangle], [10, \langle \mathbf{5}, 2\rangle, \langle +2, 5\rangle, \langle +2, 1\rangle, \langle +1, 1\rangle]$$

Where the first block is stored using 11 Bytes, and the second block is stored using 10 Bytes, and the first document identifier in each block is not delta encoded.

The performance of query processing algorithms, such as MAX-SCORE and WEAK-AND, directly depend on the ordering of documents. If documents with high probabilities of relevance, or scores, are discovered early during query processing, then tight bounds can be made, and the number of scored documents reduced. This process results in fewer scored documents, and faster query processing. It is no surprise that alternate orderings for inverted indexes have been proposed and evaluated. Arroyuelo et al. (2013) present recent research into methods of improving the compression, and decompression, of inverted indexes by re-assigning document identifiers.

Early approaches to the problem of representing sequences of unbounded, positive integers include unary, Golumb coding (Golomb, 1966), and the Elias family of codes (Elias, 1975). An example of some of these numeric compression schemes is shown in Table 2.1. The ratio between the size of the original data and the compressed representation, the compression ratio, depends on the distribution of the integers to be compressed. Therefore, an appropriate encoding scheme must be selected based on the values to be compressed.

There are two key aims in the compression of posting list data for inverted indexes; first to reduce the space requirements of the index, and second to provide rapid access to the data stored in the index. These aims imply a need for a high compression ratio, and very fast decompression algorithms. The trade-off between compression ratio and decompression rate is discussed in detail by Witten et al. (1999). Modern compression techniques such as v-byte (Williams and Zobel, 1999), Simple-9 encoding (Anh and

Table 2.1: Example of different compression schemes for inverted indexes. Note that the integer 0 does not occur in posting lists, so, it is omitted from the range of compressible integers.

Numeral	Unary	Golumb (4)	Elias γ	Elias δ	v-byte
1	1	$1,00$	1	1	$0,0000000$
2	01	$1,01$	$01,0$	$010,0$	$0,0000001$
3	001	$1,10$	$01,1$	$010,1$	$0,0000010$
4	0001	$1,11$	$001,00$	$011,00$	0.0000011
5	00001	$01,00$	$001,01$	$011,01$	$0,0000100$
6	000001	$01,01$	$001,10$	$011,10$	$0,000101$
7	0000001	$01,10$	$001,11$	$011,11$	$0,0000110$
8	00000001	$01,11$	$0001,000$	$00100,000$	$0,0000111$
9	000000001	$001,00$	$0001,001$	$00100,001$	$0,0001000$
10	0000000001	$001,01$	$0001,010$	$00100,010$	$0,0001001$

Moffat, 2005) and PForDelta encoding (Zukowski et al., 2006) can provide good balance between compression ratio and decompression speed, as observed by Arroyuelo et al. (2013).

In this thesis, we use v-byte encoding (Williams and Zobel, 1999), as it provides good compression and it can be decoded rapidly. This efficiency is partly due to the native treatment of bytes in modern computing platforms. Analysis of other integer compression schemes is considered out of scope for this thesis. However, it is reasonable to expect that our findings are generally applicable to other effective encoding schemes.

2.4 Enumeration of Terms

Enumeration of terms is a common technique for the compression of inverted indexes (Witten et al., 1999). It involves the conversion of each term-string into a term identifier. Use of this technique adds a requirement of a mapping structure, from terms to term identifiers, to enable the processing of queries.

Term enumeration can be performed during the construction of a term-level inverted index. In a monolithic setting, enumeration can be performed using two different methods. First, it can be performed at document parse-time, using a large in-memory mapping structure. Second, it can be performed at the final write stage, where each term is assigned a unique identifier as it is written to the final index structure.

Depending on the size of the vocabulary of the indexed collection, the first method could require a large amount of memory. The mapping structure must be stored in memory, as the set of terms extracted from a document must be directly converted into a sequence of term identifiers. A by-product of this method is an enumerated document collection, that may be later used to construct a number of different term dependency indexes.

The second method can be implemented by incremental enumeration. As each term is written to an inverted index of terms, the term is assigned a new number, one higher than the previous. However, this algorithm does not directly produce an enumerated version of the collection. Additionally, the size of the intermediate data, that must be sorted to collect like-terms, could be significantly larger than the size of the final index.

In a distributed processing setting, more complex methods are required to ensure that each term is assigned a unique global identifier. One approach, used by Indri,[1] is to only enumerate terms locally in index shards. Local enumeration can be implemented using similar methods to monolithic enumeration above. A problem for this method is that each query term must be converted to its enumerated value once per index shard, depending on the mapping structure, this may require an additional disk-seek, or a large amount of memory.

[1] A component of The Lemur Project, `http://www.lemurproject.org/indri.php`

A dynamic setting, where documents will be added to and removed from the index over time, adds further complexities to these term enumeration algorithms. The term mapping structure must now be extensible, and documents must be able to be efficiently translated into an enumerated form. Preferably, this enumeration process should require fewer than one random access per unique term in the document.

Investigation of systems and algorithms that efficiently perform these tasks is not within the scope of this study. For the study of term dependency indexes in this thesis, we assert that each of the collections tested have been enumerated as a pre-processing step. This ensures a fair baseline for the comparison of space requirements and efficiency of indexing algorithms. We use non-enumerated indexes for investigations on dependency model retrieval effectiveness, and on the impact of term dependency indexes on retrieval effectiveness.

2.5 Other Index Structures

The document-ordered inverted index structure has been shown to be a very effective structure for the efficiency execution of information retrieval models, for an input query. However, it is not the only structure that has been proposed.

A retrieval model requires a set of collection-level and document-level statistics for each query term, or dependency feature. So, the fundamental requirement for the execution of a retrieval model is efficient access to the required set of statistics. Here, we discuss some alternative sets of data structures that are designed to support the execution of a range of different retrieval models, using a range of different query processing algorithms. Each alternative structure is associated with different query processing algorithms, and provides different trade-offs between memory requirements, frequency of disk assesses, and space requirements. Investigation of these data structures is beyond the scope of this thesis.

2.5.1 Frequency-ordered Inverted Index

Frequency-ordered inverted indexes (Persin et al., 1996) have been proposed as a simple method of ensuring highly probably documents are observed early during query processing. As discussed by Persin et al. (1996), frequency-ordered inverted indexes permit the development of several efficient query processing algorithms. However, these algorithms are not necessarily rank safe. Rank safety is defined here as a property of an algorithm, such that the ranked list of the highest scoring documents returned by the algorithm is equivalent to the ranked list that would be returned if all documents were scored.

An additional consideration is that these data structures must be compressed using a slightly different method to the document-ordered inverted indexes discussed above (Moffat and Zobel, 1995). Document identifiers are no longer stored in increasing order, so differential encoding of document identifiers cannot be used over the entire list. However, documents that share a particular term frequency can be grouped, then documents in each frequency-group stored in increasing order. Through this modification, both the frequency values, and the list of document identifiers for each group, can be stored using differential encoding.

As an example, the following posting list shows the set of document identifiers corresponding to documents that contain the term 'term', and the frequency of the term in the document $\langle d_i, f_{t,d_i} \rangle$. Note that the posting list is shown in decreasing frequency order:

$$\text{term} \ : \langle 5,4 \rangle, \langle 1,3 \rangle, \langle 5,3 \rangle, \langle 3,1 \rangle, \langle 4,1 \rangle, \langle 9,1 \rangle, \langle 10,1 \rangle$$

Grouping frequency values, $\langle f_{t,d_i}, [d_i, d_j, \ldots] \rangle$, this posting list is represented as:

$$\text{'term'} \ : \langle 4,[5] \rangle, \langle 3,[1,5] \rangle, \langle 1,[3,4,9,10] \rangle$$

Finally, applying differential encoding to both frequency values and document identifiers:

$$\text{`term'} : \langle 4, [5] \rangle, \langle -1, [1, +4] \rangle, \langle -2, [3, +1, +5, +1] \rangle$$

The same integer compression techniques discussed for document-ordered posting lists can be applied to each of the integers in the posting list, as discussed by Moffat and Zobel (1995) and Arroyuelo et al. (2013).

A frequency-ordered inverted index is only applicable to bag-of-words retrieval models. Even if the index were modified to store positional data, it is very inefficient to locate and extract positional data for each query term in a particular document. This is because the positional data for the desired document could occur almost anywhere in each query term's posting list. Therefore, this index ordering is not considered a reasonable comparison point for the contributions in this thesis.

It is possible to consider this ordering for the full, frequent and sketch indexes of term dependencies investigated in this thesis. This modification will change the effectiveness of compression techniques, which may considerably alter the space requirements for each structure. Investigation of the efficiency of this ordering for inverted indexes of dependencies is out of scope for this thesis.

2.5.2 Impact-ordered Inverted Index

A related approach to the frequency-ordered inverted index is the impact-ordered inverted index (Anh and Moffat, 2002b). Impacts are defined as the contribution of a term for a particular document, for a given retrieval model. Anh and Moffat (2002a) define document impacts as $w_{d,t}$ in the expression:

$$S_{q,d} = \sum_{t \in q \cap d} w_{d,t} \cdot w_{q,t} \tag{2.1}$$

Where $S_{q,d}$ is the score of a document, $w_{d,t}$ is the impact of the term t for document d, and $w_{q,t}$ is the impact of the term t for the query q. Clearly, the transformation

requires that the retrieval model function can be defined as the product of these two independent factors. Metzler et al. (2008) extend this work to probabilistic retrieval models.

Generally, for retrieval models that match this pattern, impacts are real valued variables, rather than integers. Each posting will therefore will require considerably more space, than in a term frequency-based inverted index. Anh and Moffat (2002b) proposes a method of improving compression through the normalization and quantization of the impact values. The authors show that retrieval effectiveness is approximately preserved for vector space retrieval models using 5 bits or more for each quantized impact.

Similar to the frequency-ordered inverted index, the impact-ordered inverted index does not store positional data, and can not be used on its own to efficiently compute dependency retrieval models. Therefore, the structure is not considered as a comparable data structure for the dependency index data structures proposed in this thesis.

Again, it is possible to modify the full, frequent and sketch indexes of term dependencies to store impacts instead of frequency data. Note that the quantization approach used for terms may not be appropriate for term dependency scores. So, the investigation of this modification would require a study of the distributions of scores produced for each type of dependency. However, the investigation of this modification is beyond the scope of this thesis.

2.5.3 Self-Indexes

Self-indexes have recently been shown to be a remarkably effective index organization for phrase queries (Navarro and Mäkinen, 2007). This index organization is based around a set of in-memory data structures including the FM-index (Ferragina and Manzini, 2000), the compressed suffix array (Grossi and Vitter, 2000), and

the wavelet tree (Grossi et al., 2003). Recently, it has been shown that the compressed suffix array can be emulated by a compressed form of the Burrows-Wheeler Transformation (Burrows and Wheeler, 1994). This is known as the succinct suffix array (Mäkinen and Navarro, 2005, 2008), and offers improved index compression, without significantly compromising query efficiency.

These indexing structures have very attractive worst case efficiency bounds when doing "grep-like" occurrence counting in text. Fariña et al. (2012) show how to extend these indexing structures to term-based alphabets. However, the basic self-indexing framework does not directly address the *document listing problem* whereby a listing of the documents containing the search pattern in some frequency ordering is required. Muthukrishnan (2002) provided the first bounded approach to these and other related counting problems using a "document array" data structure. Subsequent research has steadily progressed the time and space efficiency of top-k document retrieval using a single search pattern (term or phrase) (Sadakane, 2007, Hon et al., 2009, Culpepper and Moffat, 2010, Patil et al., 2011, Hon et al., 2012).

Unfortunately, the body of work surrounding top-k document retrieval using self-indexes focuses primarily on singleton pattern querying in order to derive the best possible efficiency bounds, and all use character-based instead of term-based vocabulary representations which often results in indexes that are 2 to 3 times larger than the text collection being indexed, all of which must be maintained in memory. Culpepper et al. (2011) investigated the viability of using a self-indexing configuration for multi-term bag-of-words querying using a BM25 similarity computation instead of frequency-based ordering. Culpepper et al. (2012) extended these bag-of-words querying capabilities to include term-based indexes and the pre-computation enhancements of Hon et al. (2009) and Navarro and Valenzuela (2012). They show that term-based self-indexes with a variety of auxiliary data structures to support ranked document retrieval are competitive with inverted indexes in both effectiveness and efficiency.

Despite the advances in self-indexing, the approach is still hampered by two key issues: the indexes must be stored entirely in memory; and index construction requires full suffix array construction as an intermediate step. Suffix array construction is notoriously memory hungry, requiring around $9 \cdot |\mathcal{C}|$ in-memory space for large collections (Puglisi et al., 2007). In summary, self-indexing approaches for ranked document retrieval are a very promising and active area of research, but current methods are limited by the amount of physical RAM available, which translates well to only modest-sized document collections in the IR domain. A variety of in-memory inverted indexing methods also exist (Strohman and Croft, 2007, Transier and Sanders, 2010, Fontoura et al., 2011), some of which attempt to selectively include phrasal components directly within the index (Transier and Sanders, 2008). While all of these indexing approaches provide compelling efficiency gains and can be constructed using significantly less memory than current suffix-based approaches, physical memory limitations still bound the size of collection that can be supported and were therefore not considered in this dissertation.

2.5.4 Signature Indexes

Signature indexes are a index organization that was a focus of information retrieval research during the mid 1990s. In these structures, term occurrences for documents or groups of documents are stored in bitstrings, or signatures.

A signatures is defined as a sequence of w bits created to represent the data contained in each document in a collection. The signature for a document is created by hashing each term to a w string, and `OR`'ing each of these bitstrings together (Faloutsos, 1992). Queries are processed by first constructing a signature for the query, then comparing the signature of each document in the collection to the query signature. Variations on this organization have been proposed to improve query processing ef-

ficiency, including bitslices (Kent et al., 1990), and partitioning (Sacks-Davis et al., 1995).

Zobel et al. (1998) compare this type of index to inverted index data structures in depth. They demonstrate that inverted indexes require less space and process queries more efficiently than signature indexes. The contributions made in this thesis are not applicable to this type of index organization.

2.5.5 Term-Pair Indexes

To directly store statistics for some required term dependencies, some different types of term-pair indexes have been investigated in several previous studies (Broschart and Schenkel, 2012, Schenkel et al., 2007, Williams and Zobel, 1999, Williams et al., 2004, Yan et al., 2010). These index structures are all indexes of term dependencies. A common observation is that a full index of all pairs of terms makes infeasibly large requirements of disk space. Each of these structures have different approaches to reducing the space requirements.

Williams and Zobel (1999) present the next-word index. This structure is designed for the efficient processing of phrase queries of arbitrary length. This index is composed of two structures, a vocabulary component and a structure containing a set of positional posting lists. The vocabulary component stores a mapping from terms to lists of 'next' terms, each 'next' term is accompanied by an offset for the posting list for the bigram. A posting list for a bi-gram is accessed by looking up the first term, and scanning through the list of 'next' terms to find the second term. The offset stored with the second term allows direct access to the posting list for the bigram. In order to retrieve instances of longer phrases, the positional data of component bigrams are extracted and compared. This structure has been shown to significantly improve the efficiency of phrase queries (Bahle et al., 2002, Williams et al., 2004), over

positional term indexes. Bahle et al. (2002) also propose a query-log based filtering technique to discard many bi-grams from the index.

The next-word index can be considered equivalent to a positional index of 2-grams. Even though the next-word index can only store one type of term dependency, this structure is a benchmark, against which each proposed index for term dependencies proposed in this thesis will be compared.

Schenkel et al. (2007) present a similar index of small ordered-window term dependencies. This index structure is designed to store and retrieve the ordered window term dependencies used in the retrieval model proposed by Büttcher et al. (2006). This index structure stores entries for pairs of terms that occur within a small distance or window. They reduce the size of the term-pair index using two parameters; minimum score, and list length. The first restriction, min-score, requires each indexed term-pairs to produce a score higher than the threshold for some document in the collection. The second restriction restricts longer posting lists to store data for just the highest scoring documents.

More recently, Broschart and Schenkel (2012) extend this study by proposing a parameter tuning framework that optimizes either the retrieval effectiveness or the retrieval efficiency. Missing from this study is a comparison to other retrieval models, such as the bag-of-words models discussed previously. This would provide appropriate effectiveness lower bound, as it has been shown that effective bag-of-words models (Robertson and Walker, 1994, Song and Croft, 1999) can be efficiently computed over much smaller, simpler index structures (Turtle and Flood, 1995, Broder et al., 2003, Strohman et al., 2005).

Yan et al. (2010) propose a partial index of a proximity-based term dependencies. This index structure stores information for ordered term-pairs that occur with windows. In this paper, windows of width 3 or less are investigated. The authors propose two index size reduction techniques. The first method is to discard term-pair

lists where both terms are infrequent in the collection. This filter is based on the intuition that recombining these posting lists will be comparatively inexpensive. The second pruning technique is top k pruning, where only the k top documents for each term-pair posting list is retained.

To demonstrate the utility of this index structure, Yan et al. (2010) also propose a new retrieval model that uses these term proximity features. However, missing from this study is a comparison to other retrieval models for retrieval effectiveness. Further, the effect of pruning the term-pair index on retrieval effectiveness is never directly evaluated. We note that both of these filtering techniques are remarkably similar to those proposed by Schenkel et al. (2007).

Each of these pruning techniques are similar to the frequency-based filtering used by the frequent index presented in Chapter 6. Indeed, the frequent index construction algorithms presented in this thesis can be modified to be applicable to the construction of these indexes. However, the effect these filtering techniques have on retrieval effectiveness is not clear from these studies. The filtering techniques are directly tied to a specific retrieval model, meaning that tuning model parameters to optimize the retrieval effectiveness of the system may require repeated re-indexing of the data. The analysis of the effect of these filtering techniques is beyond the scope of this body of work.

2.6 Query Processing Models

A query processing model for ad-hoc information retrieval is defined as an algorithm that takes a query and a set of index structures as input, and returns the k documents most likely to be relevant, given the estimation of relevance to the query, as provided through a retrieval model.

We adopt several assumptions that are common in ad-hoc retrieval. First the query is assumed to be natural language; the query does not contain any Boolean

operators, phrase markers, or other specially defined operators or modifiers. Second, document independence is assumed, the score or probability of relevance of a document is independent of all other documents. Generally, query processing models can be divided into two models: TERM-AT-A-TIME, and DOCUMENT-AT-A-TIME. In the literature these algorithms are also called query evaluation strategies.

As discussed by Turtle and Flood (1995), a query processing algorithm can be classified as 'safe', 'rank-k-safe', or 'unsafe'. *safe* algorithms ensure that the correct ranking is produced. *rank-k-safe* ensure that the top k documents are ranked correctly. *unsafe* or *approximate* algorithms do not guarantee that any portion of the ranking is correct, however, they may provide other, weaker, guarantees. A common 'safe' evaluation optimization is to restrict the set of scored documents to documents that contain at least one query term.

The TERM-AT-A-TIME algorithm is a simple, non-optimized TERM-AT-A-TIME query processing model (Turtle and Flood, 1995). This algorithm operates by completely processing each query term in turn. It requires that a set of document score accumulators to be stored in memory for the duration of query processing. Only after all terms are processed is the final score of each document is known.

DOCUMENT-AT-A-TIME processing iterates through the set of documents, computing the score of a document entirely before proceeding to the next document. The DOCUMENT-AT-A-TIME algorithm is a simple non-optimized query processing model. This query processing model requires significantly less memory that TERM-AT-A-TIME alternatives. The identity and score of the k most relevant documents must be maintained in memory, but little else is required.

2.6.1 Term-at-a-time

Moffat and Zobel (1996) discuss how the memory requirements of the accumulators in term-at-a-time algorithms can be restricted using the `continue` and `quit`

Algorithm TERM-AT-A-TIME

```
1: for d_i ∈ 𝒟 do
2:     initialize accumulator, A_i, to 0
3: end for
4: for q_j ∈ Q do
5:     retrieve posting list iterator, 𝒫_{q_j}, for query term, q_j
6:     for d_i ∈ 𝒫_{q_j} do
7:         compute the contribution, c_{i,j}, of the term, q_j, for the document, d_i
8:         increment accumulator, A_i, by contribution, c_{i,j}
9:     end for
10: end for
11: retrieve document lengths/priors, L
12: for d_i ∈ 𝒟 do
13:     normalize accumulator, A_i, using length/prior, L_i
14: end for
15: identify the k top documents from accumulators, A
16: return return the k identified documents
```

Algorithm DOCUMENT-AT-A-TIME

```
1: initialize heap structure, H
2: retrieve document lengths/priors, L
3: for q_j ∈ 𝒬 do
4:     retrieve posting list iterator, 𝒫_{q_j}, for query term, q_j
5: end for
6: for d_i ∈ 𝒟 do
7:     initialize accumulator, A, to 0
8:     for q_j ∈ 𝒬 do
9:         compute the contribution, c_{i,j}, of the query term, q_j, for the document, d_i
10:        increment accumulator, A, by contribution, c_{i,j}
11:    end for
12:    normalize accumulator, A, using length/prior, L_i
13:    if |H| < k or min(H) < A then
14:        insert pair, (d_i, A), into heap, H
15:        if |H| > k then
16:            remove min(H) from H
17:        end if
18:    end if
19: end for
20: return  return the k documents in H
```

optimizations. These optimizations dynamically create accumulators as required, the creation of new accumulators is restricted when the current accumulator structure is larger than k. Appropriate selection of k, can ensure that this algorithm is rank-k-safe.

Other optimizations for TERM-AT-A-TIME focus on short-circuiting query evaluation. Buckley and Lewit (1985) describes an optimization that allows some terms to be omitted from processing. Their optimization tracks the difference between the score for two documents, ranked t and $(k + 1)$, where t is a parameter such that $t < k$. If the maximum possible contribution for a term is computed to be less than the difference in scores between ranks t and $k + 1$, then query processing halts, and the top k partially scored documents are returned. Turtle and Flood (1995) presents a rank-k-safe variant of this optimization.

Anh and Moffat (2006) investigates several TERM-AT-A-TIME query processing models for impact and frequency ordered inverted indexes. They propose four stages of query evaluation; `OR`, `AND`, `REFINE`, and `IGNORE`. The first stage, (`OR`), permits the creation of new accumulators the second, `AND`, restricts the creation of new accumulators, the third, `REFINE`, discards all but the top k accumulators to refine the scores for this small set of documents. The final stage, `IGNORE`, indicates that query processing can be early-terminated, as the results are rank-k-safe. Strohman and Croft (2007) extends this approach to further reduce the memory requirements for the set of accumulators.

The execution of dependency models using a TERM-AT-A-TIME algorithm on a positional index needs to compare positional data for pairs of terms, for each scored document, this is not possible for these algorithms without extensive memory requirements, or repeated reading of posting list data. Full, frequent or sketch indexes of term dependencies avoid this requirement, and can be used with this type of algo-

rithm. However, the evaluation of these structures using this type of algorithm is out of the scope of this thesis.

2.6.2 Document-at-a-time

Optimizations for the DOCUMENT-AT-A-TIME algorithm focus on avoiding the computation of scores for unlikely documents. A simple optimization is to use the posting list iterators to control the document iteration. Instead of scoring every document, only score documents that contain one or more query terms.

Turtle and Flood (1995) presents an effective optimization to this approach. The MAX-SCORE is a rank-k-safe algorithm that uses the maximum contribution for each term in the query to prune some documents from consideration. As a document is processed, the contribution for each term is added to the accumulator. After term i is scored, the maximum contribution of the remaining terms is summed and the maximum score for the document is computed. If this maximum score is lower than the k^{th} ranked document, then contributions from the remaining terms need not be computed. The query processing algorithm can proceed to the next document. Also note that this algorithm uses only the first i, highest weighted terms in the selection of the next term. The value i is determined dynamically using the k^{th} ranked document, and the maximum contribution of each term. This method ensures rank-k-safety and reduces the total number of documents considered.

Strohman et al. (2005) improve upon this method by introducing `topdocs` to the MAX-SCORE algorithm. `topdocs` is the set of highest scoring documents for a given term. This data is stored at the head of each posting list. Strohman et al. (2005) show that this data can dramatically reduce the maximum contribution for each query term, thus further reducing the number of scored documents and computation required to identify the top k documents.

When using Max-Score to execute a dependency model, the maximum contribution of each dependency feature must be pre-computed, or estimated, to allow Max-Score to operate. If positional indexes are used, then this means that all matching instances of the dependency are extracted from the intersection of positional posting lists, prior to the execution of the query. This limits the effectiveness of Max-Score. Recent research has investigated methods of improving estimations of the maximum contribution of dependency features (Macdonald et al., 2011a,b). The indexes of term dependencies investigated in this thesis directly store the collection statistics for each indexed term dependency, allowing for efficient, accurate maximum contribution computation.

The Weak-And query processing model (Broder et al., 2003) is an alternative Document-at-a-Time query processing model. The Weak-And operates by iterating through the set of documents using a measurement similar to the maximum score. Upper bounds on the contribution for each term are used to determine documents that may meet the current threshold, minimum score for candidacy. Depending on the threshold parameter settings, this query processing model can be used in a rank-k-safe or a unsafe method.

In the context of dependency retrieval models, the computation of the upper bound for dependency features is an expensive operation. If the upper bound is overestimated, the algorithm will not be able to prune documents from consideration, it will be forced to check all documents that contain one or more query terms. Broder et al. (2003) enable phrase operators by modifying the Weak-And algorithm to never check the upper bound of these features. The contributions of this thesis, direct indexes of term dependencies, enable the inclusion of phrase operators in the Weak-And algorithm, and may permit further computational savings.

2.6.3 Multi-pass algorithms

Wang et al. (2010) present a multi-pass learning-to-rank approach to query processing model efficiency. This method uses a learned model to predict the cost of processing each query feature. The first pass through the collection scores documents using only the most efficient query features (such as just the query terms). Repeated passes over the returned documents accumulate scores from more complex and costly features, and further reduce the set of documents.

We note that this algorithm requires collection statistics to estimate the cost of processing a particular feature. Where positional indexes are used, there is a circular dependency for dependency features, the positional data must be processed to determine the collection statistics, that will then determine the cost of processing the positional data.

Full, frequent, and sketch indexes of term dependencies, as analyzed in this thesis, would reduce the cost of some valuable term dependency query features, allowing these features to contribute to the scores produced by the initial pass. Further, these structure would provide the collection statistics that enables the estimation of the cost of processing each feature. These indexes may help reduce the number of passes by allowing some more complex features to be processed in earlier passes. Secondarily, this approach may allow a further reduction in the size of the document set for subsequent passes.

2.7 Cache Structures

Caching mechanisms have been studied in almost all areas of computer science as methods of reducing processing time and improving throughput. Caching generally involves optimizing the location of frequently accessed data items, usually to faster levels of storage. The aim of a cache is to ensure frequent, or soon-to-be accessed data items are stored in high-speed, but low-capacity storage (e.g. RAM), while

infrequently accessed data items remain in low-speed and high-capacity storage (e.g. hard disks). This type of optimization can also be applied to storing frequently computed values.

In this thesis, we analyze different index structures in order to improve the efficiency of dependency models. Cache structures are a traditional method of achieving this goal. However, a lack of widely available large query logs makes the evaluation of these structures infeasible in this thesis. Note that the query log index, investigated in Chapter 7, is equivalent to a large cache structure of posting lists. In future work, we intend to extend this analysis, to determine the hit-rates, and the costs of cache misses for a number of these structures, when supporting the execution of dependency models.

Ozcan et al. (2012) propose a five level hierarchy for search engine caching. The five levels consist of a top k snippet cache, a top k result cache, an intersected posting list cache, a posting list cache, and a document cache. While posting list caches and document caches, the lowest two levels, must be stored in memory to improve efficiency, cache structures at each of the three higher levels can be stored on disk and still improve efficiency (Cambazoglu et al., 2010).

2.7.1 Top k Cache Structures

Cache structures for the top k results has been an active area of research for large scale information retrieval systems. Baeza-Yates and Saint-Jean (2003) propose a static, memory-based cache structure that stores results for the most frequent queries in the query log. This paper demonstrated that around 20% of submitted queries can be satisfied by this cache structure alone. A dynamic approach to caching is proposed by Markatos (2001). Dynamic caches, in this context, store results for recently submitted queries. Their cache is designed as an in-memory structure of query-response pairs. Several cache ejection policies are investigated to limit the memory require-

ments of the dynamic cache, including variations on LRU (least recently used), and FBR (frequency-based replacement) policies.

Fagni et al. (2006) show that a hybrid approach can improve the hit rate over either dynamic and static caches. This hybrid approach consists of both a static set of cache entries and a dynamically maintained cache. This research shows that the contents of a top-k cache should include both recently queried data, and data that is frequently observed in the query log. Gan and Suel (2009) present a method of directly integrating these desirable properties. They introduce a feature based cache replacement algorithm, where the features capture both the historical frequency of a query, its recent usage and several other desirable properties.

To the best of our knowledge, Cambazoglu et al. (2010) are the first to propose that disk-based data structures could be an efficient option for top-k caches. Authors observe that large amounts of query processing time is saved, even when the top-k results must be retrieved from disk. Furthermore, the authors assert that cache structures stored on disk for web scale systems are not directly limited by space requirements, so it is possible to have 'infinite' caches. Cambazoglu et al. (2010) assert that ejection policies should only focus on removing invalid or obsolete data from the cache. They assert that cached results should only be made invalid as new documents are crawled and added to the index. To allow cache entries to be discarded in an efficient manner, they propose and investigate top-k caches involving time-to-live or a 'freshness' feature for each cache entry. This feature records the last time that this query was evaluated by the search engine. Older entries are considered invalid, and the entry can be overwritten or reused.

An important observation made in several related papers is that the use of top-k caching systems can dramatically change the distribution of a query stream (Baeza-Yates et al., 2007). This observation influences each of the query log analyses made in this thesis. If a top-k cache is used, then only a subset of unique queries in the

stream must be processed by the search engine. However, with cache ejection policies, a number of queries will need to be repeatedly executed.

A key concern in live systems is staleness. In a dynamic setting, documents are continually added to the index. If one of these newly added documents is more relevant than a document stored in the cache for a particular query, then the cache entry is stale. A common approach is to have time-based ejection policies, where each cache entry is ejected after a predefined time. This approach introduces a trade-off between additional query processing and the risk of returning a stale cache entry in response to a query.

Haahr et al. (2009) present a method of avoiding the need for time-based cached ejection policies in this dynamic context. In order to ensure that a cache entry does not become stale, as documents are added to the index, the cache is also inspected, and updated with the new documents, as necessary. This approach ensures that cache entries do not become stale, with respect to a dynamically updating index structure. So, the cache ejection policies in this system can now focus on the relative frequency of each query in the query stream.

2.7.2 Posting List Cache Structures

Posting list caches have also been demonstrated as a effective method of improving information retrieval efficiency. This type of cache stores replicas of disk-based posting list data in memory for efficient access. This type of cache has the potential to produce a much higher hit rate than top k caches. Research into this type of cache has generally been through multi-level caching systems, that is, a system that incorporates more than one cache level. For example, both Saraiva et al. (2001) and Baeza-Yates et al. (2007) investigate two-level cache systems. In both studies, a top k result cache and a posting list cache are used to improve retrieval efficiency. They show that the addition of a posting list cache improves efficiency over a system using just a top-k

cache. While the hit rate of a top-k query cache is limited to between 20% and 30%, a perfect posting list cache has the potential to achieve hit rates over 90%.

Caches of intersections of posting list data have also been shown to have the potential to improve information retrieval efficiency. To the best of our knowledge, there are only three studies to date that investigate this type of cache (Long and Suel, 2005, Marin et al., 2010, Ozcan et al., 2012). Long and Suel (2005) extend two-level caching to three-level caching. Their work allows result lists, posting lists, and intersected posting lists to be stored in the cache. In their system, cached intersected lists are stored on disk. They are able to show significant improvement over two-level caching schemes. Marin et al. (2010) investigate implementations of the three-level cache structure in distributed computing environments. They show that this three-level cache structure is more efficient and more scalable than a two-level structure. Ozcan et al. (2012) extend the three-level cache structure to a five-level structure (as discussed above). Interestingly, they show that the hit rate of a cache of intersected lists is very similar to the hit rate of posting lists. Their data shows that query processing times and hit rates are improved using a five-level structure over two- or three-level structures, given similar space requirements.

However, these three-level systems and the corresponding analyses are strictly limited to the processing of conjunctive `AND`-based retrieval models. A second problem is omission, in each of these studies, of the structure of the intersected list cache. That is, there is no analysis of the construction or maintenance of this data structure in each of these papers. Finally, the cost of querying the cache for the presence or absence for each possible intersected posting list is not mentioned in any of these studies. For pair-wise list intersections, this cost may be exponential in the length of the query.

CHAPTER 3

DATA AND METRICS

3.1 Collections

This study uses data from TREC collections for the majority of the experiments. Primarily, we use the Robust-04, GOV2, and ClueWeb-09 collections.[1] Clueweb-09 consists of two named sub-collections, Category-A (Cat-A) and Category-B (Cat-B), where Cat-B is a subset containing approximately 10%of Cat-A. In accordance with a study of spam document in the collection by Cormack et al. (2010), we filter the Clueweb-09 collection to the set of documents that are in the 60^{th} percentile of the Fusion Spam scores. These spam scores were generated using the Waterloo spam classifier (Cormack et al., 2010), and are distributed with the collection.

General statistics for each of these these collections are shown in Table 3.1. These collections provide a wide range of collection sizes, allowing us to test the scalability of the construction of each index data structure. The collections span several different

[1] http://trec.nist.gov/data/

Table 3.1: Statistics for each corpus that will be used in this thesis. Note that disk space in this table is compressed using GZIP.

Collection	Documents	Vocabulary	# of Terms	Disk Space
Robust-04	528,155	656,586	252,013,235	583 MB
GOV2	25,205,179	35,996,686	22,332,601,362	80 GB
ClueWeb-09-Cat-B	33,836,981	39,693,640	26,070,670,503	145 GB
ClueWeb-09-Cat-A	201,404,339	113,554,643	123,612,811,042	715 GB

Table 3.2: Statistics for query logs used in this thesis. MQT is the TREC Million Query Track.

Name	TREC Year	# of Queries	Vocab.	Avg.Length
AOL	2006	20.4 M	2.8 M	2.80 terms
MSN	2006	14.7 M	1.5 M	2.78 terms
MQT, GOV2	2007/8	20,000	15,882	4.65
MQT, Clueweb	2009	40,000	26,704	2.52

types of documents, including; newswire data, domain specific web documents (`.gov`), and general web documents.

3.2 Queries for Retrieval Efficiency Experiments

In order to quantify any retrieval efficiency improvements that can be achieved using term dependency indexes, we require a large and diverse sample of appropriate queries to execute. In this thesis we use three sources of queries, AOL and MSN query logs, and queries collected for and used in the TREC Million Query Track (MQT). Statistics for each of these sets of queries is shown in Table 3.2.

We perform several normalizing functions as a preprocessing step for all queries in each of these logs. This includes case normalization, punctuation removal, and acronym normalization.

3.3 Queries for Retrieval Effectiveness Experiments

The Robust-04, GOV2, and Clueweb-09 collections have each been used in several TREC tracks, and many publications use them as standard information retrieval collections. Through the TREC conferences we have access to a commonly available set of topics and relevance judgments for each collection. We display a short summary of this data in Table 3.1.

The topic sets we use in this body of work are accumulated over a number of TREC Tracks. All 250 Robust-04 topics were used in the 2004 TREC Robust track. 200 of

Table 3.3: Statistics for each set of topics that will be used to evaluate retrieval effectiveness in this thesis.

Name	# of Topics	Vocab. Size	Avg. Length	# of Judgments
Robust-04, Titles	250	555	2.7	311,410
Robust-04, Descs.	250	1273	8.2	311,410
GOV2, Titles	150	403	2.9	135,352
GOV2, Descs.	150	676	6.1	135,352
Clueweb-09, Titles	150	400	2.2	84,366
Clueweb-09, Descs	150	715	4.6	84,366

these Robust-04 topics were also used in previous TREC tracks. The 150 GOV2 topics were accumulated from the 2004, 2005, and 2006 TREC terabyte tracks. Finally, the 150 Clueweb-09 topics were collected from the 2009 to 2011 TREC web tracks. Each topic has two forms, a short title and a longer description. Only title topics were used in the official TREC competitions.

Stopwords and some select Stop Structures (Huston and Croft, 2010) have been removed from these topics. We use the INQUERY stopword set (Callan et al., 1994). Stop Structures were manually identified, non-content phrases repeatedly used in topic descriptions, for example "find documents that discuss" and "retrieve documents regarding".

3.4 Retrieval Effectiveness Metrics

We measure retrieval effectiveness using three frequently used retrieval metrics; mean average precision (MAP), normalized discounted cumulative gain at rank 20 (nDCG@20), and precision at 20 (P@20). Each of these metrics are defined over a set of query rankings, where each document in each ranking is annotated with a relevance judgment. Statistical differences are computed using Fisher's randomization test, with statistical differences reported where $\alpha = 0.05$.

Relevance is defined in this thesis as topical relevance. A document is considered relevant if it answers some significant part of the information need represented by the

query. Specifically, a document is consider relevant for the input query if a human annotator manually judges the document to be relevant.

Relevance judgments created for the TREC tracks used in this thesis all used a judgment pooling method. A range of different information retrieval systems are used to provide between 50 and 100 documents to a pool, for each topic. All documents in the pool are then annotated by human assessors.

Relevance can be either a binary or a multi-valued classification. Binary classification indicates that each document is either relevant to the information need, or it is not. Multivalued classification allows documents to be marked on a scale indicating the degree to which the document is relevant to the information need. If there is no relevance judgment for a document, it is assumed not to be relevant to the information need.

Precision at k (P@k) is a binary retrieval effectiveness measure. It is defined as the fraction of the top k ranked documents that are relevant, for the input query. When P@k is computed over a set of queries, the mean per-query P@k is returned. P@k a single query is defined:

$$P@k(q) = \frac{|R_q \cap D_{q,k}|}{k} \tag{3.1}$$

Where R_q is the set of relevant documents for the query q, and $D_{q,k}$ is the set of retrieved documents ranked above k.

Mean average precision (MAP) is also a binary retrieval effectiveness measure. It is defined over a set of queries, it is the mean of the average precision (AP) measurements for each query, in a set of queries. Average precision is approximately the area under a precision-recall curve. More specifically, it is defined as:

$$AP(q) = \frac{1}{|R_q|} \sum_{i=1}^{k} \text{P@i(q)} \cdot I(d_k \in R_q) \tag{3.2}$$

Where R_q is the set of relevant documents for the query q, P@k is defined above, and $I(d_k \in R_q)$ is an indicator function that returns 1 if the document at rank k is in the set of relevant documents. k is defined as the maximum rank considered, unless otherwise specified, it is assumed to be 1000. Using this function we can define MAP:

$$MAP = \frac{\sum_{q \in Q} AP(q)}{|Q|} \tag{3.3}$$

Normalized discounted cumulative gain @ k (nDCG@k) is a graded retrieval effectiveness metric. This metric relates the rank of each document to its graded utility, or relevance. The metric is normalized using an optimal ranking, where the relevant documents are ranked in decreasing order of relevance above all non relevant documents. The unnormalized discounted cumulative gain (DCG@k) is defined as:

$$DCG@k = r_{q,1} + \sum_{i=2}^{k} \frac{r_{q,i}}{\log_2(i)} \tag{3.4}$$

Where $r_{q,i}$ is the relevance score of the document at rank i, for query q. Using this function we can define nDCG@k:

$$nDCG@k = \frac{DCG@k}{IDCG@k} \tag{3.5}$$

Where IDCG@k is DCG@k computed optimal or ideal ranking of documents. A ideal ranking is defined for a query, q, where $r_{q,i} >= r_{q,i+1}$, for all ranks, i.

Fisher's randomization test will be used throughout this thesis to detect statistically significant differences between retrieval models. This choice follows the observations of Smucker et al. (2007). Fisher's randomization test makes the null hypothesis that A and system B are identical. It operates by permuting the per-query evaluations (by randomly swapping the observations between the two systems), and comparing

the aggregate difference between each permutation and the original, observed systems. The null hypothesis is rejected where a small fraction (α) of the generated permutations exhibit larger differences than the original systems, as measured using a specific metric. Throughout this thesis we set the threshold for rejecting the null hypothesis at $\alpha = 0.05$.

3.5 Scalability

A key requirement of any index data structure is that it can be constructed in a scalable manner. For an index construction algorithm to be considered scalable in a monolithic computing environment, the space, time and memory requirements should not grow excessively as the size of the input data increases.

An inverted index construction algorithm can be viewed as a variant of a large scale sorting algorithm. It takes as input a sequence of terms, as observed in documents or other retrieval units, and produces as output a data structure that collects instances of like-terms in posting lists. It follows that index construction algorithms that require at most $O(|C| \log |C|)$ execution time are considered scalable, where $|C|$ is a measure of collection size. In this thesis, we assert a static memory bound, as the collection size is increased, when empirically testing the scalability of different indexing algorithms.

Given that memory is a physical requirement, we expect that memory requirements should be parameter controllable. The algorithm has a minimum requirement, but additional memory can be used to improve the efficiency of the algorithm. We assert that an indexing algorithm can only be considered scalable if sub-linear growth rates in minimum memory requirements are observed.

Disk space requirements can be divided into two categories for indexing algorithms; intermediate space requirements, and final space requirements of the index structure. For an algorithm to be considered scalable, intermediate disk space requirements of the indexing algorithm should not exceed the final disk space requirements of the

index structure. For the final disk space requirements to be considered scalable, the index structure should grow at most linearly with the size of the collection.

Note that the size of the collection should be measured in the number of terms or term dependencies in the collection. The number of term dependencies extracted from each document depends on the type of dependency being extracted. For example; there are $(|D|-7)\cdot 7+\sum_{1\leq i<7} i$ unordered windows of width 8, containing 2 words, in a document containing $|D|$ words. More generally, using unordered windows of width w, containing n words, the total number of windows extracted from a document of length $|D|$ is:

$$(|D|-(w-1))\cdot\binom{(w-1)}{(n-1)}+\sum_{1\leq i<(w-1)}\binom{i}{(n-1)}$$

In the context of a parallel processing environment, the definition of scalable is slightly different; as the problem size is increased and the number of processing machines is increased by a matching proportion, a distributed implementation of the algorithm executes in time that remains fixed, or grows only slowly. More precisely, if a collection of size $|C|$ can be indexed on a single processor in time t, then a problem of size $p\cdot|C|$, when distributed over p machines, should be solvable in time not significantly greater than t. Moffat and Zobel (2004) discuss issues of performance evaluation in distributed environments.

CHAPTER 4

COMPARISON OF DEPENDENCY MODELS

4.1 Introduction

As outlined in Chapter 1, in this thesis we aim to investigate methods of improving the efficiency of dependency models, in particular the sequential dependence model (SDM), and the weighted sequential dependence model (WSDM). However, there is little value in improving the efficiency of these models, if there are simpler, more efficient models that are equally effective. In this chapter, we compare a wide range of different dependency models to determine the most effective ones. Additionally, we evaluate the relative benefits from many-term dependencies.

Recent research demonstrated that retrieval models incorporating term dependencies (dependency models) (Bendersky et al., 2010, Lv and Zhai, 2009, Metzler and Croft, 2005, Peng et al., 2007, Song et al., 2008) can consistently outperform benchmark "bag-of-words" models (Amati and Van Rijsbergen, 2002, Ponte and Croft, 1998, Robertson and Walker, 1994) over a variety of collections. As defined in Chapter 1, a dependency model is any model that exploits potential relationships between two or more words to improve a document ranking.

Although there have been a number of comparisons of dependency models to various different bag-of-words baselines, there has been surprisingly little comparison between these models. Given the importance of dependency models, it is critical to provide comparisons and baselines that can be used to establish the effectiveness of new models, instead of showing an improvement compared to relatively weak base-

lines. In this chapter, we provide these comparisons and a detailed study of the effectiveness of different types of proximity features.

In the first part of the chapter, we describe a number of state-of-the-art dependency models that use features based on the proximity of pairs of terms (bi-terms). Each of these retrieval models was previously introduced in Chapter 2. For this comparison, we use a range of TREC collections, including both short (title) and long (description) queries. By using these collections, query sets, and open source software, our results can be easily reproduced and used as baselines or benchmarks in future studies. The parameters for each model are extensively tuned to maximize performance, and 5-fold cross validation is used to avoid over-fitting. Details of parameter settings and query folds are provided in Appendix A.

Some recently proposed dependency models use proximity features involving more than two terms (Bendersky and Croft, 2009, Song et al., 2008, Svore et al., 2010, Tao and Zhai, 2007). We define a many-term dependency as a set of three of more terms that are assumed to be dependent. The studies comparing many-term dependency features to bi-term features have been inconclusive. The second part of this chapter provides more comprehensive evidence of the relative effectiveness of these proximity features. We compare the best bi-term dependency models to both existing many-term dependency models and to several new many-term dependency models. The new models are created by adding many-term features to two effective bi-term dependency models, the sequential dependence model (Metzler and Croft, 2005) and the weighted sequential dependence model (Bendersky et al., 2010). Similar to the first part of the chapter, the parameters for each model are extensively tuned and 5-fold cross validation is used.

To constrain the scope of this study, we place several restrictions on the type of proximity-based dependency models that we investigate in this chapter and the thesis. These restrictions are designed to ensure fair comparisons between the mod-

els. First, we base the comparison on ad-hoc information retrieval and limit the scope of the study to retrieval models that do not use external data sources, such as Wikipedia. That is, we do not make use of data sources external to the target collection to select features or determine parameters for features. We also do not assume internal document structures, such as document fields or hyperlinks between documents. We restrict the process of selection or generation of term dependencies from the input query so that it does not rely on external information, collection statistics, or a pseudo-relevance feedback algorithm (Croft et al., 2010). These restrictions ensure that each tested model has access to identical information, and computing resources, thus allowing the direct attribution of retrieval effectiveness improvements or degradations to differences in model formulation, and specifically, the features used. Further, these restrictions make these models widely applicable in many different information retrieval problems.

Results from our experiments show that the performance of dependency models can be improved significantly through appropriate parameter tuning. This may not be a new or surprising conclusion, but the extent of the improvements possible is quite noticeable, and there are many published results where this tuning does not appear to have been done (Armstrong et al., 2009a,b). We also confirm the previous results showing that bi-term dependency features consistently improve effectiveness compared to bag-of-words models. The comparison between the bi-term dependency models shows that the variant of the weighted sequential dependence model (Bendersky et al., 2010) tested in this study exhibits consistently strong performance across all collections and query types. In regards to the comparison between short and long queries, we observe that dependency features have more potential to improve longer queries than shorter queries. We also observe that many-term proximity features have the potential to improve retrieval effectiveness over the strongest bi-term dependency

models. However, more research, and probably more training data, is required to fully exploit these features.

The major contributions presented in the chapter are:

- a systematic comparison of bi-term dependency models,
- a comparison of the effectiveness of bi-term dependency models across short and long queries,
- new many-term dependency models based on bi-term dependency models,
- a systematic comparison of many-term dependency models,
- tuned parameter settings for each tested retrieval model, for three standard information retrieval collections and query sets.

4.2 Dependency Models

4.2.1 Bi-term Dependency Models

4.2.1.1 Language Modeling

The language modeling framework for information retrieval (Ponte and Croft, 1998) has been shown to be an effective bag-of-words retrieval model. This model allows for various assumptions in the estimation of the probability of relevance. Query likelihood (QL) (Ponte and Croft, 1998) is commonly used as a strong bag-of-words baseline retrieval model. This model ranks documents using a Dirichlet-smoothed language model:

$$P(D_j|Q) \stackrel{rank}{=} \sum_{q_i \in Q} \frac{tf_{q_i,D_j} + \mu \cdot \frac{cf_{q_i}}{|C|}}{|D_j| + \mu}$$

Each document D_j, in the collection C, is evaluated for the query Q. μ is the smoothing parameter.

Several different methods have been proposed to model dependencies between terms in this framework. Metzler and Croft (2005) propose the Markov random field retrieval model for term dependencies. This framework explicitly models the relationships between query terms. The sequential dependence model (SDM) assumes that all pairs of sequential terms extracted from the query are dependent. It models each bi-term dependency using two types of proximity features; an ordered and an unordered window. The ordered windows matches n-grams or phrases in each evaluated document, the unordered window matches each pair of terms that occur in a window of 8 terms or less. SDM scores each term, ordered window and unordered window feature using a smoothed language modeling estimate. A weighted linear combination is then used to produce a final estimate of the probability of relevance for a document, given the input query.

$$
\begin{aligned}
P(D_j|Q) &\overset{rank}{=} \sum_{c \in C(Q)} \lambda_C f_C(c, D_j) \\
&= \sum_{c \in T} \lambda_T f_T(c, D_j) + \sum_{c \in O} \lambda_O f_O(c, D_j) + \sum_{c \in U} \lambda_U f_U(c, D_j) \\
T &= \{q_i \in Q\} \\
O &= U = \{(q_i, q_{i+1}) \in Q\} \\
f_T(c, D_j) &= \log P(q_i|D_j) \\
f_O(c, D_j) &= \log P(\#od1(q_i, q_{i+1})|D_j) \\
f_U(c, D_j) &= \log P(\#uw8(q_i, q_{i+1})|D_j) \\
P(x|D_j) &= \frac{tf_{x,D_j} + \mu \frac{tf_x}{|C|}}{|D_j| + \mu}
\end{aligned}
$$

Where μ is the Dirichlet smoothing parameter, and the combination of features is controlled by three parameters, λ_T, λ_O and λ_U. Additionally, $\#od1$ is an ordered window operator, and $uw8$ is the unordered window operator.

Table 4.1: Feature functions used by the WSDM-Internal retrieval model.

Feature	Description
c^1	Constant value for terms (1.0)
$cf^1(t)$	Collection frequency of term t
$df^1(t)$	Document frequency of term t
c^2	Constant value for bi-terms (1.0)
$cf^2(\#od1(t_1, t_2))$	Collection frequency of bi-term; t_1, t_2
$df^2(\#od1(t_1, t_2))$	Document frequency of bi-term; t_1, t_2

Bendersky et al. (2010) extend SDM to incorporate term and window specific weights (WSDM). Each term and window is assigned a weight, λ, using a weighted linear combination of features, g, extracted for the term or window.

$$
\begin{aligned}
P(Q|D_j) &\overset{\text{rank}}{=} \sum_{c \in C(Q)} \lambda_C f_C(c) \\
&= \sum_{c \in T} \lambda_T(c) f_T(c, D_j) + \sum_{c \in O} \lambda_O(c) f_O(c, D_j) + \sum_{c \in U} \lambda_U(c) f_U(c, D_j) \\
T &= \{q_i \in Q\} \\
O &= U = \{(q_i, q_{i+1}) \in Q\} \\
f_T(c, D_j) &= \log P(q_i|D_j) \\
f_O(c, D_j) &= \log P(\#od1(q_i, q_{i+1})|D_j) \\
f_U(c, D_j) &= \log P(\#uw8(q_i, q_{i+1})|D_j) \\
P(x|D) &= \frac{tf_{x,D} + \mu \frac{tf_x}{|C|}}{|D| + \mu} \\
\lambda_T(q_i) &= \sum_{j \in K_1} w_j^1 \cdot g_j^1(q_i) \\
\lambda_O(q_i, q_{i+1}) = \lambda_U(q_i, q_{i+1}) &= \sum_{j \in K_2} w_j^2 \cdot g_j^2(q_i, q_{i+1})
\end{aligned}
$$

Where μ is the smoothing parameter, and the parameters to the model are the feature weights, w_j. In their initial study, a total of 18 features are used to estimate the optimal weight for each term and window. In later studies, the feature set is reduced

to 13 features without exhibiting diminished performance (Bendersky et al., 2012). Several of these features are computed over external data sources, including a query log, Wikipedia and the Google n-grams collection. In accordance with our restrictions, we limit this model to the set of features that are computed over the target collection only. We label this model variant WSDM-Internal (WSDM-Int). The subset of features used to estimate the weight of each term and bi-term feature are listed in Table 4.1.

4.2.1.2 Divergence from Randomness

Retrieval models based on the divergence from randomness (DFR) framework (Amati and Van Rijsbergen, 2002) have been used in a number of studies. Term dependencies have recently been introduced to this framework (Peng et al., 2007). Similar to SDM, the proximity divergence from randomness model (pDFR) assumes that all adjacent pairs of terms are dependent. In their formulation, terms are scored using the PL2 scoring model, and bi-terms are scored using the BiL2 scoring model. The score for each document is the weighted sum of the term and bi-term components. The authors state that other DFR models can be used to score each component, which we intend to investigate in future work.

In this study, we test two pDFR models. First, we investigate the model proposed by Peng et al. (2007) that uses PL2 to score unigrams, and BiL2 to score bigrams (pDFR-Bil2). We also investigate a variation that uses PL2 to score both unigrams and bigrams (pDFR-PL2). The scoring function of each of these models is defined mathematically in Appendix A.

4.2.1.3 BM25

BM25 (Robertson and Walker, 1994) is an effective extension of the Binary Independence Model (Robertson and Jones, 1976) (BIM). It is important to note that BIM makes a strong assumption of term independence (Cooper, 1991). Specifically,

the theoretical underpinnings of BIM, and therefore BM25, preclude the inclusion of term dependency information into the estimation of the probability of relevance. Even so, several heuristic retrieval models that combine BM25 components with term proximity-based features have been proposed.

Rasolofo and Savoy (2003) present an version of the BM25 model that includes term dependency features. Their model (BM25-TP) extends the BM25 model with a feature that is proportional to the inverse square of the distance between each pair of queried terms, up to a maximum distance of 5. Büttcher et al. (2006) investigate the optimization of this function for efficient retrieval. Svore et al. (2010) further investigates this model in comparison to other BM25-based dependency models. We note that, as proposed, BM25-TP uses all possible term pairs extracted from the query. This causes an exponential growth of execution time as the size of the query grows, making the model infeasible for longer queries. In this study, we use the variation of BM25-TP proposed by Svore et al. (2010) that only assumes that all sequential pairs of query terms are dependent. The scoring function of this model is defined mathematically in Appendix A.

We also include a variant of this retrieval model (BM25-TP2), also proposed by Svore et al. (2010). This model is similar to BM25-TP, except that the score for each bi-term feature is no longer proportional to the inverse squared distance between each matched term pair. Instead, the feature is scored according to the number of matching instances using a BM25-like function. Again, the scoring function of this model is defined mathematically in Appendix A.

4.2.2 Many-term dependencies

Only a small number of many-term dependency models have been proposed. This, in itself, is is considered evidence that many-term dependencies may be less effective than bi-term dependencies. In this chapter, we investigate two recently proposed

models; BM25-Span, and the positional language model (PLM). We also propose several new many-term dependency models by augmenting SDM and WSDM-Int with many-term dependency features.

Song et al. (2008) propose the BM25-Span model. This model assumes that all queried terms are dependent. They model this single set of dependent terms by grouping term instances in a given document into *spans*. The BM25-Span model scores the detected spans using a modified BM25 scoring function. The BM25-Span scoring function evaluates each span by interpolating between the span width and number of query terms in each span.

The positional language model (PLM) was proposed by Lv and Zhai (2009). Similar to BM25-Span, this model assumes that all queried terms are dependent on each other. PLM operates by propagating each occurrence of each query term in the document to neighboring locations. The document is then scored at each location, or term position, in the document. Kernel functions are used to determine the exact propagated frequency of each term in the document to the specific position to be scored. Several methods are used for producing an aggregate score from the individual position scores.

In this chapter, we consider the two best performing variants proposed in the original study; the best-position strategy, using the Gaussian kernel (PLM), and the multi-σ strategy, using two gaussian kernels (PLM-2). The best-position strategy scores each document using the score from the maximum scoring position in the document. The multi-σ interpolates between two best-position strategies, with different kernel parameters. As suggested by Lv and Zhai (2009), the second kernel in our PLM-2 implementation is a whole document language model, $\sigma_2 = \infty$. We note that this particular kernel is equivalent to the query likelihood model (QL). Both models are implemented using Dirichlet smoothing.

Table 4.2: Descriptions of the potential functions used in various extensions to SDM. Unordered window widths are held constant at 4 times term count (Metzler and Croft, 2005).

Feature	Description
Uni	Unigram feature
O2 / U2	Ordered / Unordered window of pairs of terms
O3 / U3	Ordered / Unordered window of sets of three terms
O4 / U4	Ordered / Unordered window of sets of four terms

The scoring functions of BM25-Span, PLM and PLM-2 are all defined mathematically in Appendix A.

We now propose several new many-term dependency models. We start by extending SDM (Metzler and Croft, 2005) to include many-term dependencies. As mentioned previously, SDM is a variant of the Markov random field model that uses three types of potential functions; terms, ordered windows and unordered windows.

We extend this model by adding new potential functions to the model that evaluate larger sets of dependent terms. Table 4.2 lists each of the potential functions in the extended model. Each function is computed in a similar manner to the functions in the original SDM. Note that the width of each unordered window feature is set at four times the number of terms, as in the original study (Metzler and Croft, 2005). Various many-term dependency models are constructed by selecting different subsets of features. For example, the `Uni+O23+U23` model includes the unigram function; two bi-term functions (O2, and U2); and two 3-term functions (O3, and U3). In these models `O` indicates an ordered window is used, and `U` indicates an unordered window is used. In each model, each function is associated with one weight parameter. All features are computed as smoothed language model estimates, using Dirichlet smoothing. Similar to SDM, for an input query, each potential function is populated with all sequential sets of terms extracted from the query.

Table 4.3: Feature functions used by the WSDM-internal-3 retrieval model, in addition to the parameters shown in Table 4.1.

Feature	Description
c^3	Constant value for 3 term dependencies (1.0)
$cf^3(\#1(t_1, t_2, t_3))$	Collection frequency of trigram; t_1, t_2, t_3
$df^3(\#1(t_1, t_2, t_3))$	Document frequency of trigram; t_1, t_2, t_3

We note that each of these features is also present in the full dependence model (FDM) (Metzler and Croft, 2005). However, in FDM all ordered windows are weighted with the parameter λ_O, and similarly, all unordered windows are weighted the parameter λ_U.

We also construct a variant of the WSDM-Int model that incorporates three-term dependencies (WSDM-Int-3). WSDM-Int-3 extends WSDM-Int by including three-term features, similar to the existing two-term features. The weight for each three-term dependency is determined using a linear combination of the features in Table 4.3. Term and bi-term features are weighted as in WSDM-Int. This extension adds three new parameters to the WSDM-Int model, one for each three-term weighting feature. Again, the scoring functions of each of these many-term variant models are defined mathematically in Appendix A.

4.3 Experiments

4.3.1 Experimental Framework

To test the effectiveness of dependency retrieval models, we use three TREC collections, Robust04, GOV2, and ClueWeb-09-Cat-B. See Chapter 3 for more details on each collection. Indexed terms are stemmed using the Porter 2 stemmer.[1] All retrieval models are implemented using the Galago Search Engine.[2]

[1] http://snowball.tartarus.org/algorithms/english/stemmer.html

[2] A component of The Lemur Project, http://www.lemurproject.org/galago.php

Given the limited number of queries for each collection, 5-fold cross-validation is used to minimize over-fitting without reducing the number of learning instances. Topics for each collection are randomly divided into 5 folds. The parameters for each model are tuned on 4-of-5 folds. The final fold in each case is used to evaluate the optimal parameters. This process is repeated 5 times, once for each fold. For each fold, parameters are tuned using a coordinate ascent algorithm (Metzler, 2007), using 10 random restarts. Mean average precision (MAP) is the optimized metric for all retrieval models.

As discussed in Chapter 3, for each collection, each type of query, and each retrieval model, we report the mean average precision (MAP), normalized discounted cumulative gain at rank 20 (nDCG@20), and precision at rank 20 (P@20). For each collection, each metric is computed over the joint results, as combined from the 5 test folds. Statistical differences between models are computed using the Fisher randomization test as suggested by Smucker et al. (2007), where $\alpha = 0.05$.

The full details of the query folds, learned parameters, fold-level evaluation metrics, and p-values for statistical tests is documented in Appendix A.

4.3.2 Parameter Tuning

Most of the models investigated in this chapter were originally proposed and tested over a variety of different collections and queries, and any parameters suggested in each corresponding study are unlikely to be optimal for other collections. In this section, we present a case study on tuning the SDM parameters. Metzler and Croft (2005) suggest that the SDM parameters, $(\lambda_T, \lambda_O, \lambda_U)$, should be set to $(0.85, 0.1, 0.05)$. The smoothing parameter μ is assumed to be $2,500$, the default smoothing parameter in the canonical implementation. [3].

[3] A component of The Lemur Project, `http://www.lemurproject.org/indri.php`

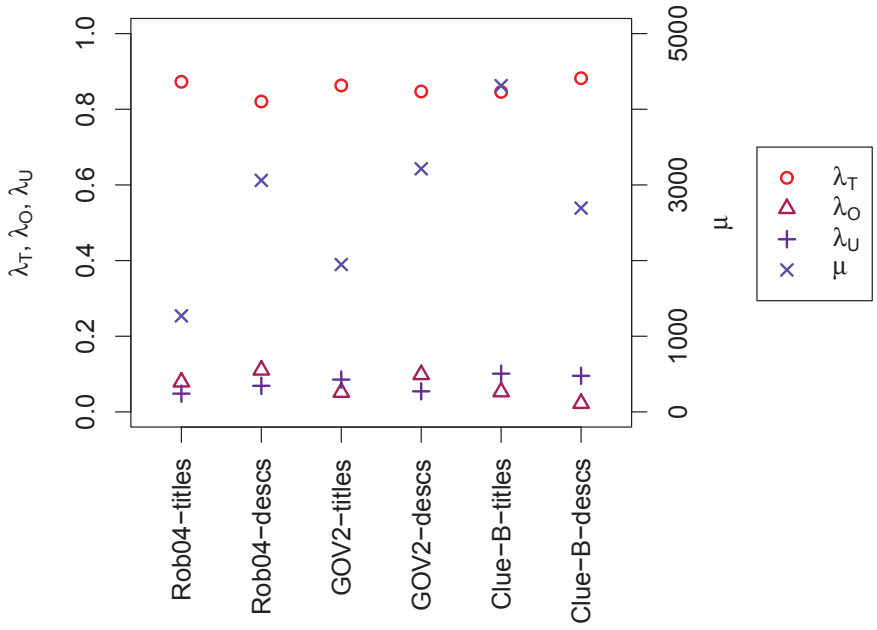


Figure 4.1: Average parameter settings across the 5 tuned folds for each collection, and each type of query. Left axis indicates values for each of the λ parameters, right axis indicates values for μ.

Table 4.4: Comparison of default to tuned parameter settings for SDM. Significance testing is performed between default and tuned results using the Fisher randomization test ($\alpha = 0.05$). Significant improvement over default parameters is indicated^{+}.

Collection	Model	MAP	nDCG@20	P@20
Robust-04, Title	SDM Default	0.264	0.424	0.374
	SDM Tuned	0.263	0.423	0.372
Robust-04, Desc.	SDM Default	0.255	0.404	0.345
	SDM Tuned	0.258	0.406	0.349
GOV2, Title	SDM Default	0.320	0.441	0.546
	SDM Tuned	0.326^{+}	0.449	0.557
GOV2, Desc.	SDM Default	0.273	0.413	0.506
	SDM Tuned	0.283^{+}	0.414	0.518^{+}
Clueweb-09-B, Title	SDM Default	0.103	0.224	0.320
	SDM Tuned	0.108^{+}	0.239^{+}	0.343^{+}
Clueweb-09-B, Desc.	SDM Default	0.076	0.180	0.226
	SDM Tuned	0.078	0.200^{+}	0.255^{+}

The 5-fold cross-validation method used in this chapter learns 5 settings for each parameter, one setting per test fold. The average learned parameter settings for each collection, and type of query is shown in Figure 4.1. This graph suggests that, with the exception of μ, the SDM parameters are relatively stable. The optimal parameters are similar for all collections and query types, and the parameters are close to the suggested default parameters. Further, we observe very low standard deviation ($\sigma_p < 0.02$) across query folds, for each parameter, p, and each collection and query set. The optimal setting for μ appears to grow with the number of query terms, and the size of the average document. However, the impact that a significant change in μ has on the final ranking is much smaller than a similar change in any of the other three parameters. These observations imply that the learned parameters are stable, and that optimal parameters for one collection could be used effectively on another collection.

We next investigate the retrieval performance differences between default and tuned parameters. Table 4.4 shows the retrieval performance of the default parameter settings, and the tuned parameter settings. Results over the smallest collection, Robust-04, do not change significantly after tuning the model parameters. This is likely to be because the original study presented results over some subsets of this collection. However, appropriate tuning of model parameters results in significant improvements in effectiveness for the larger GOV2 and Clueweb-09-Cat-B collections.

We make similar observations for all tested retrieval models that have suggested parameter settings. These observations demonstrate that even small changes in parameter values can have a significant effect on retrieval model effectiveness. Further, it is clear that if the suggested parameter values are naïvely used in a benchmark retrieval model, its performance may be significantly diminished. Reduced or low performance for a benchmark method is likely to produce erroneous conclusions about

Table 4.5: Significant differences, using the MAP metric, between bi-term dependency models and SDM as the baseline. Significance is computed using the Fisher randomization test ($\alpha = 0.05$). The first letter of each model is used to indicates a significant improvement over the paired model, '−' indicates no significant difference is observed.

	Robust-04		GOV2		Clueweb-09-B	
Models	Titles	Desc.	Titles	Desc.	Titles	Desc.
SDM / BM25-TP	−	S	−	S	−	−
SDM / BM25-TP-2	S	S	S	S	−	−
SDM / pDFR-BiL2	S	S	S	S	S	−
SDM / pDFR-PL2	−	S	S	S	P	−
SDM / WSDM-Int	W	W	−	W	W	−

significant improvements for a proposed model. While this is not a new observation (Armstrong et al., 2009b), it deserves restating.

4.3.3 Comparison of Bi-Term Dependency Models

The section presents results from the systematic comparison of bi-term dependency models. This comparison allows us to determine the relative effectiveness of the different dependency models that have been proposed and to provide a strong benchmark to investigate the utility of many-term dependencies. Recall that each model is evaluated using 5-fold cross validation, with each fold tuned using a coordinate ascent algorithm (Metzler, 2007), for each collection, and each type of query.

A summary of results is displayed in Table 4.5. This table shows significant differences, for the MAP metric, between each of the bi-term dependency models and SDM as the baseline model. Significant differences are evaluated using the Fisher randomization test and indicated in each table ($\alpha = 0.05$). Details of optimized results, as aggregated across query folds, for each model, each collection, and each type of query, are shown in Table 4.6. Significant improvement or degradation, relative to the query likelihood (QL), is indicated in this table.

Table 4.6: Comparison of dependency models over Robust-04, GOV2, and Clueweb-09-Cat-B collections. Significant improvement or degradation with respect to the query likelihood model (QL) is indicated ($^+/^-$).

	Robust-04 collection					
	Topic titles			Topic descriptions		
Model	MAP	nDCG@20	P@20	MAP	nDCG@20	P@20
QL	0.252	0.412	0.365	0.244	0.389	0.334
BM25	0.254	0.412	0.363	0.237^-	0.390	0.331
PL2	0.253	0.418^+	0.369	0.229^-	0.389	0.329
BM25-TP	0.262^+	0.418	0.371	0.243	0.394	0.336
BM25-TP-2	0.248	0.396	0.348^-	0.215^-	0.356^-	0.302^-
pDFR-BiL2	0.258^+	0.422^+	0.372	0.234^-	0.393	0.335
pDFR-PL2	0.260^+	0.422^+	0.375^+	0.235	0.393	0.333
SDM	0.263^+	0.423^+	0.375^+	0.258^+	0.406^+	0.349^+
WSDM-Int	0.269^+	0.432^+	0.382^+	0.278^+	0.428^+	0.365^+
	GOV2 collection					
	Topic titles			Topic descriptions		
Model	MAP	nDCG@20	P@20	MAP	nDCG@20	P@20
QL	0.298	0.413	0.511	0.257	0.378	0.472
BM25	0.299	0.435^+	0.530^+	0.261	0.401^+	0.484
PL2	0.300	0.415	0.516	0.258	0.390^+	0.479
BM25-TP	0.321^+	0.445^+	0.556^+	0.272^+	0.407^+	0.510^+
BM25-TP-2	0.273^-	0.382	0.480	0.250	0.392	0.495
pDFR-BiL2	0.313^+	0.437^+	0.536^+	0.266^+	0.394^+	0.486
pDFR-PL2	0.317^+	0.441^+	0.544^+	0.270^+	0.403^+	0.494^+
SDM	0.326^+	0.449^+	0.557^+	0.283^+	0.414^+	0.518^+
WSDM-Int	0.329^+	0.450^+	0.556^+	0.298^+	0.425^+	0.533^+
	Clueweb-09-Cat-B collection					
	Topic titles			Topic descriptions		
Model	MAP	nDCG@20	P@20	MAP	nDCG@20	P@20
QL	0.098	0.221	0.321	0.074	0.189	0.244
BM25	0.099	0.223	0.324	0.081^+	0.201	0.260
PL2	0.105^+	0.233^+	0.337^+	0.077^+	0.194	0.247
BM25-TP	0.109^+	0.242^+	0.349^+	0.084^+	0.201	0.258
BM25-TP-2	0.104	0.244^+	0.342^+	0.078	0.195	0.254
pDFR-BiL2	0.102^+	0.225	0.326	0.076	0.192	0.245
pDFR-PL2	0.111^+	0.248^+	0.358^+	0.080^+	0.193	0.244
SDM	0.108^+	0.239^+	0.343^+	0.078	0.200	0.255
WSDM-Int	0.113^+	0.245^+	0.354^+	0.083^+	0.199	0.255

It is clear from this data that WSDM-Int is the most consistently effective bi-term dependency model. It significantly outperforms all other retrieval models in several settings. We also note that SDM is a very strong retrieval model, it significantly outperforms several other models on the Robust-04 and GOV2 collections, as is shown in Table 4.5. This data also shows that WSDM-Int significantly outperforms SDM in several settings.

All other bi-term retrieval models show some significant improved performance, relative to the QL baseline. Interestingly, the BM25-TP2 model does not perform well, even showing significant degradation of performance in some cases. This may be because this model does not control the contribution of bi-term features using a parameter, resulting in the score contribution of bi-terms is being overvalued, relative to the contribution of terms.

These results also confirm previously published findings; bi-term models can consistently improve information retrieval on the Robust-04 and GOV2 collections. Interestingly, the significant improvements observed on the Robust-04 and GOV2 collections, are much lower on the Clueweb-09-Cat-B collection. The relatively low performance improvements with dependency models using the current Clueweb-09 queries has also been observed at TREC and in recent publications (Bendersky et al., 2012, Raiber and Kurland, 2013). As more queries are developed for this corpus, we plan to study this issue further.

When comparing the performance of short and long queries, we observe that the use of bi-term proximity dependencies produces much larger improvements for description topics, than for title topics, for the Robust-04 and GOV2 collections. One obvious cause for this is that many more bi-term dependencies are extracted from the longer description topics. We also observe that for the Robust-04 collection, the effectiveness of the best performing model on description topics, (WSDM-Int), is significantly better than the effectiveness with title topics, using the MAP metric.

Table 4.7: Investigation of many-term proximity features over the Robust-04, and GOV2 collections. Significant improvements and degradation with respect to SDM are indicated ($^{+}/^{-}$).

	Robust-04, Topic desc.			GOV2, Topic desc.		
Model	MAP	nDCG@20	P@20	MAP	nDCG@20	P@20
SDM	0.258	0.406	0.349	0.283	0.414	0.518
WSDM-Int	0.278^{+}	0.428^{+}	0.365^{+}	0.298^{+}	0.425^{+}	0.533^{+}
BM25-Span	0.243^{-}	0.394^{-}	0.333^{-}	0.261^{-}	0.401	0.484^{-}
PLM	0.250^{-}	0.386^{-}	0.332^{-}	0.247^{-}	0.337^{-}	0.433^{-}
PLM-2	0.260	0.398	0.345	0.276	0.390^{-}	0.485^{-}
Uni+O234	0.258	0.409	0.351	0.279^{-}	0.407^{-}	0.511
Uni+O234+U2	0.259^{+}	0.408	0.351	0.283	0.409^{-}	0.516
Uni+O23+U23	0.258	0.411^{+}	0.353	0.281	0.409	0.516
Uni+O234+U234	0.259	0.409	0.353	0.282	0.410	0.511
WSDM-Int-3	0.280^{+}	0.428^{+}	0.364^{+}	0.297^{+}	0.425	0.531

	Clueweb-09-B, Topic desc.		
Model	MAP	nDCG@20	P@20
SDM	0.078	0.200	0.255
WSDM-Int	0.083	0.199	0.255
BM25-Span	0.085^{+}	0.205	0.261
PLM	0.064^{-}	0.153^{-}	0.198^{-}
PLM-2	0.075	0.190	0.239^{-}
Uni+O234	0.074^{-}	0.190^{-}	0.245^{-}
Uni+O234+U2	0.079	0.202	0.256
Uni+O23+U23	0.076	0.196	0.252
Uni+O234+U234	0.079	0.199	0.253
WSDM-Int-3	0.080	0.202	0.252

For this collection, WSDM-Int is able to extract more informative features from the long queries compared to the associated short queries.

4.3.4 Many-term Proximity Features

Based on the results from the comparison of bi-term dependency models, we select SDM and WSDM-Int as benchmark methods against which to evaluate the benefit of many-term dependencies for retrieval effectiveness. We evaluate this by investigating three existing many-term dependency retrieval models; BM25-Spans, PLM, and PLM-2. We also construct new many-term dependency models, by modifying

Table 4.8: Significance differences between pairs of dependency models, using MAP metric. Significance is computed using the Fisher randomization test ($\alpha = 0.05$). The first letter of each model is used to indicates a significant improvement over the paired model, '–' indicates no significant difference is observed.

	Robust-04	GOV2	Clueweb-09-B
Models	Desc.	Desc.	Desc.
WSDM-Int / BM25-Span	W	W	–
WSDM-Int / PLM	W	W	W
WSDM-Int / PLM-2	W	W	W
WSDM-Int / Uni+O234	W	W	W
WSDM-Int / Uni+O234+U2	W	W	–
WSDM-Int / Uni+O23+U23	W	W	–
WSDM-Int / Uni+O234+U234	W	W	–
WSDM-Int / WSDM-Int-3	W3	–	–

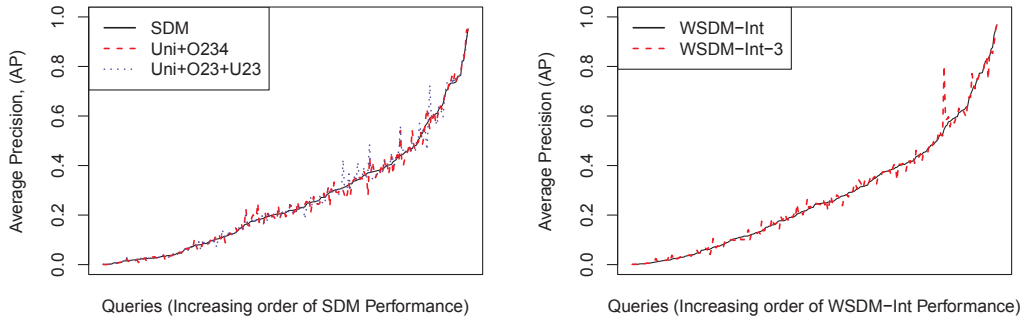


Figure 4.2: Per-query average precision (AP) for Robust-04, topic descriptions, using SDM and 2 related many-term proximity models.

the SDM and WSDM-Int retrieval models. We construct SDM variants that use dependent sets of three and four terms, as described in Section 4.2. Finally, we construct a variant of WSDM-Int, WSDM-Int-3 which includes three-term dependencies. Note that for the extended SDM and WSDM-Int models, all term dependencies are extracted sequentially from the query.

Similar to the comparison of bi-term models, each many-term dependency model is evaluated using 5-fold cross validation, where each fold is tuned using a coordinate ascent algorithm (Metzler, 2007). We focus on topic descriptions in this section, as

title queries are frequently too short to extract dependent sets of more than two terms.

Note that PLM and PLM-2 are only tuned for the Robust-04 collection. To evaluate a given document for these retrieval models, for a particular query, each position in the document must be evaluated, and the maximum score is returned. In the worst case, every position in the collection must be scored independently. To reduce this overhead, we implement a simulated annealing algorithm to reduce the number of positions tested to locate the maximum scoring position in the document. This algorithm reduced the time to evaluate queries by a factor of 20, without any measurable change in retrieval effectiveness. However, even with this efficiency optimization, and the document-length estimation optimization presented by Lv and Zhai (2009), tuning the parameters of this retrieval model using coordinate ascent over larger collections remains infeasible. In lieu of tuned optimal parameters, we report results for GOV2, and Clueweb-09-Cat-B using the optimal parameters from Robust-04.

The results from these experiments are displayed in Table 4.7. Significant differences are indicated in the table with respect to the SDM benchmark. We also show significance testing with respect to the WSDM-Int benchmark in Table 4.8.

We observe that many-term dependency models do not consistently improve retrieval performance over bi-term models. Small improvements can be observed in some cases. WSDM-Int-3, in particular, shows significant improvements over benchmark models for the Robust-04 collection. Investigation of the performance of WSDM-Int-3 for the GOV2 and Clueweb-09-Cat-B collections shows some evidence that the model is overfitting to the 4 folds of training data, thereby reducing performance on the test fold. However, in general, many-term features do not improve aggregate retrieval metrics with respect to the benchmark bi-term dependency models.

Figure 4.2 shows two per-query result graphs, one comparing SDM with two variant many-term models, and one comparing WSDM-Int to WSDM-Int-3. Data shown

is only for the topic descriptions from the Robust-04 collection. Similar trends were observed for the other two collections.

This data shows that each of these models improves and degrades different queries, resulting in similar average performance. It also suggests that many-term proximity features may still be able to significantly improve retrieval performance. However, optimizing the use of many-term dependencies may require a more selective approach to the generation of sets of dependent terms, or to the selection of model features for an input query. Further, the availability of more training data, such as a large query click log, would be likely to make a significant difference to our results.

4.4 Preliminary Results for Future Work

4.4.1 Model Stability and Parameter Variance

An important feature of some retrieval models is portability. This problem is commonly faced by enterprise or personal search, where systems must be deployed and retrieve information from unseen collections. Importantly, model parameters can not be directly tuned for these situations. One part of this problem is measuring the portability or stability of each dependency model for a variety of different sizes and types of collections.

An important first step in this investigation is to investigate changes in optimal parameter settings across a range of collections, as has been generated as part of the above comparisons. Table 4.9 shows the mean, standard deviation, and coefficient of variance, or the ratio between the standard deviation and the mean, for each parameter for each model. These statistics are computed as an aggregate across all three collections, both types of queries, and all folds.

We observe that parameters for dependency models generally exhibit a higher coefficient of variance than the bag-of-words models. An exception to this observation, SDM show very little variance across the three collections. However, as observed in

Table 4.9: Aggregate statistics of learnt parameters across all training folds, collections, and topic titles and descriptions, for bag-of-words models, and for bi-term dependency models. CV is the coefficient of variance.

Bag-Of-Words Models

Parameter	Mean	Std. Dev.	CV ($\frac{\sigma}{\mu}$)
QL			
μ	1877.755	559.046	0.298
PL2			
c	7.766	4.877	0.628
BM25			
b	0.369	0.121	0.327
k	1.982	2.020	1.019

Bi-Term Models

BM25-TP			
b	0.286	0.116	0.406
k	1.732	1.539	0.889
BM25-TP2			
b	0.196	0.098	0.502
k	2.570	2.274	0.885
pDFR-BiL2			
c	7.396	5.003	0.677
c_p	899.027	1286.990	1.432
λ	1.303	0.225	0.173
pDFR-PL2			
c	17.188	23.872	1.389
c_p	16.357	20.392	1.247
λ	0.885	0.032	0.036
SDM			
μ	2749.430	1138.803	0.414
λ_T	0.855	0.032	0.037
λ_O	0.069	0.033	0.470
λ_U	0.076	0.032	0.419
WSDM-Int			
μ	2429.847	910.163	0.375
λ_{c^1}	0.812	0.121	0.149
λ_{1-df}	-0.044	0.021	-0.484
λ_{1-cf}	0.004	0.015	3.657
λ_{c^2}	0.044	0.023	0.535
λ_{2-df}	0.000	0.003	15.384
λ_{2-cf}	0.001	0.004	2.389

Many-Term Models

BM25-Span			
b	0.279	0.124	0.444
k	1.804	2.275	1.261
λ	0.902	0.364	0.403
γ	0.267	0.064	0.238
PLM			
μ	1847.277	1301.687	0.705
σ	488.423	676.370	1.385
PLM-2			
λ	0.446	0.062	0.138
μ	1450.059	507.823	0.350
σ	8.613	15.262	1.772
Uni+O23+U23			
μ	3477.464	1544.513	0.444
λ_U	0.823	0.059	0.071
λ_{O2}	0.065	0.024	0.377
λ_{O3}	0.019	0.038	1.979
λ_{U2}	0.055	0.035	0.628
λ_{U3}	0.037	0.023	0.624
WSDM-Int-3			
μ	2797.431	1284.671	0.459
λ_{c^1}	0.752	0.151	0.200
λ_{1-df}	-0.057	0.029	-0.514
λ_{1-cf}	0.019	0.018	0.945
λ_{c^2}	0.045	0.018	0.408
λ_{2-df}	-0.002	0.002	-0.829
λ_{2-cf}	0.003	0.003	1.122
λ_{c^3}	0.010	0.018	1.855
λ_{3-df}	0.001	0.003	3.359
λ_{3-cf}	0.004	0.009	2.533

Table 4.10: Comparison of the performance of WSDM-Int and WSDM (Bendersky et al., 2012). $^+$ indicates a significant improvement over WSDM-Int.

Collection	Model	MAP	nDCG@20	P@20	ERR@20
Robust-04, Title	WSDM-Int	0.269	0.432	0.382	0.119
	WSDM	0.271^+	0.435^+	0.383	0.119
Robust-04, Desc.	WSDM-Int	0.278	0.428	0.365	0.123
	WSDM	0.283^+	0.431	0.366	0.125^+
GOV2, Title	WSDM-Int	0.329	0.450	0.556	0.176
	WSDM	0.331^+	0.453	0.563^+	0.176
GOV2, Desc.	WSDM-Int	0.298	0.425	0.533	0.167
	WSDM	0.303^+	0.426	0.536	0.165
Clueweb-09-B, Title	WSDM-Int	0.113	0.245	0.354	0.130
	WSDM	0.111	0.244	0.352	0.132
Clueweb-09-B, Desc.	WSDM-Int	0.083	0.199	0.255	0.118
	WSDM	0.088^+	0.219^+	0.283^+	0.125^+

Section 4.3.2, even small changes in parameter settings can have significant changes in the retrieval effectiveness. So, we can see that variance in optimal parameter settings is not a measure of portability or stability. The next step of this study is to investigate if there is a connection between the variance of optimal parameter settings and changes in retrieval effectiveness, for a range of collections.

Additionally, a future direction for this work is to investigate the optimization of retrieval model settings for unseen collections. This will include training retrieval models on one or more TREC collections, then testing the learned parameters on a held-out collection.

4.4.2 External Data Sources

In this section, we show some initial research into the relaxation of the restriction against external data sources. There is some evidence that external data sources can help significantly improve ad-hoc retrieval performance. For example, 6 of the 7 top performing models at the TREC 2012 Web Track use external data sources.

As originally proposed, WSDM (Bendersky et al., 2010) is a bi-term dependency model that uses three external data sources; Wikipedia titles, the MSN query log, and Google n-grams. This model uses these features to determine term and window specific weights. By design, WSDM allows the inclusion of arbitrary external features into the weighting of each term and window, making this model appropriate for further investigation of any benefit of external data sources for dependency models.

Table 4.10 shows a comparison between WSDM, and one of the strongest performing bi-term dependency models, as determined in this chapter, WSDM-Int. We test both models for each collection and query type, where both models are tuned using 5 fold cross validation. We observe that the external data sources used in WSDM can significantly improve the performance of this model. However, the improvements are relatively small. Future work should focus on the isolation and identification of the external features that provide the largest benefits for this model, and investigate alternative data sources that may provide larger gains in retrieval performance.

4.5 Summary

In this study, we performed a systematic comparison of state-of-the-art bi-term dependency models. We proposed new many-term dependency models, based on bi-term dependency models. We performed a systematic comparison of many-term dependency models, using the strongest performing bi-term models as benchmark models. Additionally, we provided tuned parameters for a wide range of popular dependency models, for three standard test collections.

The retrieval models investigated here are subject to several restrictions in the selection of retrieval models. We restricted the comparison to models that use proximity-based dependencies between sequentially extracted sets of queried terms, that do not require external data sources, and do not require the use of pseudo-relevance feedback algorithms.

Our results support previous findings that bi-term dependency models can consistently outperform bag-of-words models. We observe that dependency models produce the largest improvements over bag-of-words models on longer queries. The best performing bi-term model, given the restrictions applied, is a variant of the weighted sequential dependence model. Our experiments also show that many-term dependency models do not consistently outperform bi-term models. However, per-query analysis shows that many-term proximity features have some potential to improve retrieval performance, if used in a more selective manner.

There are three extensions to this study that could be conducted in future work. First, it is important to investigate of model stability across changes in the target collection. Second, an investigation of the actual benefits from external data sources for dependency models, as compared to the strongest baselines, is vital for improving ad-hoc information retrieval. We provide some initial experiments on both of these topics in Section 4.4.

Finally, the results in Section 4.3.4 indicate that alternative methods of extracting many-term dependencies from queries may further improve retrieval performance. This study would include the use of linguistic and external data sources to identify and extract term dependencies between queried terms and also within sets of expansion terms. Several effective models that use these data sources for this task have recently been proposed (Aktolga et al., 2011, Bendersky and Croft, 2008, Maxwell and Croft, 2013, Park et al., 2011, Xue and Croft, 2010). This extended comparison should include the comparison of these models to the top performing dependency models identified here.

CHAPTER 5

TERM DEPENDENCY INDEXES

5.1 Introduction

In this chapter, we discuss existing indexing data structures and index construction algorithms that can store and retrieve proximity dependency statistics. We perform experiments that investigate the properties of each of these structures.

In Chapter 4, we observed that the sequential dependence model (SDM) (Metzler and Croft, 2005) and the internal variant of the weighted sequential dependence model (WSDM-Internal) (Bendersky et al., 2010). In this Chapter, we focus on the constructing indexes that store statistics for the window features that are used by both SDM and WSDM-Int. Both types of windows were originally defined by Metzler and Croft (2005).

Both ordered and unordered windows have two parameters: the width, and the number of terms. In this thesis, we use a simple nomenclature for windows, an ordered window of width X, containing Y terms is denoted `od-wX-nY`. Similarly, an unordered window of width X, containing Y terms is denoted `uw-wX-nY`. The fundamental difference between the two types of windows is that in an ordered window, the terms must occur in the specified order, whereas in an unordered window the terms are not required to occur in any particular order.

Figure 5.1 illustrates an example instance of each type of window for the terms: t_1, t_2 and t_3. It is important to note that the width parameter is defined differently for each type of window. In a valid instance of an ordered window, the distance between each adjacent pairs of terms must be less than the specified width. In the example,

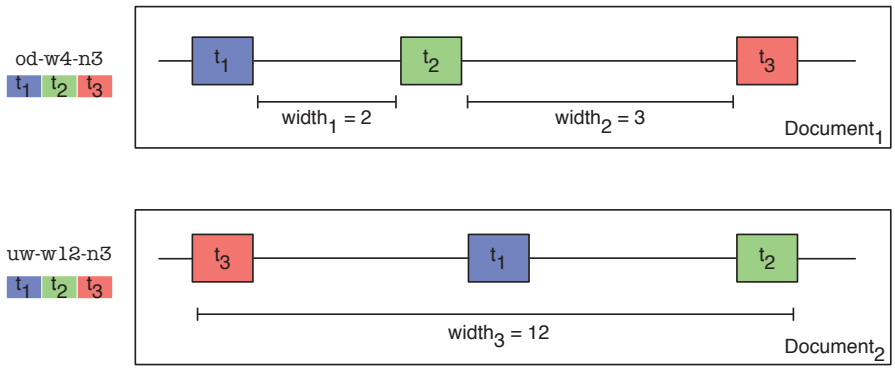


Figure 5.1: Example instances of ordered and unordered window. Note that "width" has different meanings for ordered windows and unordered windows.

we can see that both the distance between t_1 and t_2, and the distance between t_2, and t_3 are less than the specified width, 4. Width for unordered windows defines the maximum width of a span of text that includes all query terms. In the example, we can see that all terms occur within a window of width, 12.

The main contributions of this chapter include:

- discussion of existing index structures for term dependencies;
- analysis of the differences between three different algorithms for detecting windows using positional posting lists; and
- analysis of the distribution of window dependencies in three English Collections.

5.2 Index Data Structures

In order to execute a bag-of-words model, such as query likelihood (Ponte and Croft, 1998), BM25 (Robertson and Walker, 1994), and PL2 (Amati and Van Rijsbergen, 2002), an index must provide the following statistics:

- $|C|$; collection length in terms;
- $|D_i|$; length of the document i in terms;
- cf_t; the frequency of term t in the collection;

- df_t; the number of documents that contain term t;
- $tf_{t,i}$; the frequency of term t in document i;

The strongest performing bi-term dependency model from Chapter 4, WSDM-Int (Bendersky et al., 2012), extends this list of required statistics with the following:

- $cf_{\#od1(t_i,t_j)}$; the collection frequency of ordered windows of width 1, that contains terms t_i and t_j;
- $df_{\#od1(t_i,t_j)}$; the number of documents that contain an instance of an ordered window of width 1, that contains terms t_i and t_j;
- $cf_{\#uw8(t_i,t_j)}$; the collection frequency of unordered windows of width 8, that contains terms t_i and t_j;
- $df_{\#uw8(t_i,t_j)}$; the number of documents that contain an instance of an unordered window of width 8, that contains terms t_i and t_j;
- $tf_{\#od1(t_i,t_j),i}$; the frequency of ordered windows of width 1, that contains terms, t_i and t_j, in document i;
- $tf_{\#uw8(t_i,t_j),i}$; and, the frequency of unordered windows of width 8, that contains terms, t_i and t_j,in document i.

WSDM-Int-3, further extends this list of required statistics with collection- and document-level statistics of ordered and unordered windows containing three terms.

This chapter discusses and compares some existing approaches to storing and extracting term, bi-term and many-term statistics. Specifically, we focus on inverted indexes of ordered and unordered windows, as the models that use these features, SDM, WSDM-Int, WSDM-Int-3, have demonstrated very strong retrieval performance. Other retrieval models also use these types of features, see the discussion in Chapter 2.

To execute queries efficiently, a search engine requires a set of data structures that can efficiently retrieve or generate each of the required statistics for the execution of an input query. The storage and retrieval of term and window frequency statistics, $tf_{X,i}$ and $df_{X,i}$, is commonly supported by an inverted index. We will discuss this structure at length in the following sections.

While, the collection length, $|C|$, and the length of each document, $|D_i|$, are not the focus of this chapter, it is important to indicate how they may be stored efficiently. Each of these values can be stored in memory, or on disk. Assuming document identifiers are enumerated sequentially, document length values can be stored in a simple array of integers, requiring $\mathcal{O}(|\mathcal{D}| \cdot \log(max_i(|\mathcal{D}_i|)))$ bits, where $\mathcal{D}$ is the set of all documents in the collection, C, and $|\mathcal{D}_i|$ is the length of the i^{th} document in terms. This array can be stored in memory, even for large collections. For example, storing the lengths of 500 million documents, (e.g. Clueweb-09-Cat-A), would require almost 2 GB of memory space, assuming a 4 Byte integer is used to store each length. If document identifier enumeration is not sequential, or if the collection contains very large numbers of small documents (e.g. microblog entries), it may be necessary to store this data on disk, and read the data using a buffered stream.

5.3 Index Data Structures for Term Dependencies

5.3.1 Full Inverted Indexes

Perhaps the simplest solution to indexing windows for term dependencies is to construct an inverted index of every window instance in the collection. This structure maps each unique window instance in the collection to a posting list of document frequencies. Using this structure, the window statistics required by the corresponding dependency model can be directly extracted from this index:

$$\#window(t_i, t_j) \rightarrow cf_{\#window(t_i,t_j)};\ df_{\#window(t_i,t_j)};\ [\ \langle d_i,\ tf_{\#window(t_i,t_j),d_i}\ \rangle, \ldots,\ ]$$

An example inverted index of `od-w1-n2` is shown here:

$$
\begin{aligned}
&\vdots \\
&\text{`}a\,a' \rightarrow 10; 4; \langle 1, 3, \rangle, \langle 2, 2\rangle, \langle 5, 2\rangle, \langle 6, 3\rangle, \ldots \\
&\text{`}a\,b' \rightarrow 11; 3; \langle 1, 5, \rangle, \langle 3, 2\rangle, \langle 5, 4\rangle, \ldots \\
&\text{`}a\,c' \rightarrow 3; 1; \langle 7, 3\rangle, \ldots \\
&\vdots
\end{aligned}
$$

A major advantage of this structure is retrieval efficiency. At query time, the posting list for an indexed term dependency can be extracted, and directly used in the processing of each document.

However, this type of index can have very large space requirements. There are several factors that will affect the space requirements. The number of postings, the size of the vocabulary, number of bytes required to store each vocabulary item, will each affect the space requirements.

We investigate the distributions and vocabulary growth rates for different types of ordered and unordered windows in Section 5.6.2. The vocabulary and average posting list sizes for indexes of ordered and unordered windows, for a range of widths and numbers of terms, can be estimated from this data. This allows us to predict the space requirements of full indexes of ordered and unordered windows.

5.3.2 Positional Inverted Indexes

An important variation on the inverted index is the inverted positional index, or positional index (Witten et al., 1999). This structure stores a mapping from each term to a posting list that contains each document the term occurs in, as well as a list of positions at which the term occurs for each document. A positional posting list for term t is defined here as:

$$t \rightarrow cf_t;\, df_t;\, \Big[\, \langle D_i,\, tf_{t,D_i},\, [p_0,\, p_1, \ldots,\, p_{f_{t,D_i}-1}] \,\rangle, \ldots, \,\Big]$$

Where p_j is the j^{th} term extracted from document D_i. All other variables in this example are defined above. See Section 2.3 for a discussion of the compression techniques that can be applied to this structure.

An example inverted positional index of terms is shown here:

$$\begin{aligned}
&\vdots \\
&\text{`}a' \rightarrow 5; 3; \langle 1, 1, [3] \rangle, \langle 2, 2[2, 10] \rangle, \langle 5, 2[3, 8] \rangle \\
&\text{`}b' \rightarrow 7; 3; \langle 1, 2, [4, 5] \rangle, \langle 3, 3[1, 6, 9] \rangle, \langle 5, 2[2, 6] \rangle \\
&\vdots
\end{aligned}$$

Unlike the full index, term dependency statistics must be re-computed from positional data at query time, reducing retrieval efficiency dramatically. However, positional data allows the reconstruction of a wide variety of term dependency features at query time, without the cost of constructing a dedicated index. Using the positions from the above example, we extract instances of `od-w1-n2`, and `uw-w8-n2` that each contain terms a and b:

$$\begin{aligned}
&\texttt{od-w1-n2}(a, b) \rightarrow \langle 1, [3\text{-}4] \rangle \\
&\texttt{uw-w8-n2}(a, b) \rightarrow \langle 1, [3\text{-}4, 3\text{-}5] \rangle, \langle 5, [2\text{-}3, 2\text{-}8, 3\text{-}6, 6\text{-}8] \rangle
\end{aligned}$$

Several of the displayed unordered windows reuse term instances. This observation leads to a efficiency-effectiveness trade-offs for the extraction of ordered and unordered windows. Several different algorithms can be devised depending on a series of choices on whether or not to reuse a specific term-instance in multiple window instances. The above example shows the output if all term instances are reused in all

Algorithm UNORDERED-WINDOWS-ALL

```
1: INPUT: PostingIterator[ ] itrs : One postings list iterator per term.
2: INPUT: width : unordered window parameter.
3: OUTPUT : WindowArray output : Output array of windows
4: itr_0 = itrs.pop()
5: while not done_0 do
6:     output.addAll( ExtractWindows(itrs, pos_0, pos_0)
7:     itr_0.next()
8: end while
9: return output;
10: function ExtractWindows(itrs, begin, end)    ▷ Recursive function to find windows
11:     itr_i = itrs.pop()
12:     while not done_i do
13:         newBegin = min(currBegin, itr_i.currentPosition())
14:         newEnd = max(currEnd, itr_i.currentPosition())
15:         if newEnd − newBegin ≤ width then
16:             if itrs is empty then
17:                 output.add(newBegin, newEnd)
18:             else
19:                 output.addAll( ExtractWindows(itrs, newBegin, newEnd)
20:             end if
21:         end if
22:         itr_i.next()
23:     end while
24:     itr_i.reset()
25:     itrs.push(itr_i)
26:     return output;
27: end function
```

possible windows. The UNORDERED-WINDOWS-ALL algorithm details an algorithm that generates this output. In this algorithm, each list of positions must be processed several times to ensure that all windows are extracted. So, the worst case complexity of this algorithm is bounded at $\mathcal{O}(\prod_i^n |itr_i|)$ operations.

Alternatively, observing that each extracted instance covers a specific region of the document, we could omit dominated windows. A window dominates another when both share a starting position, but its ending position is smaller than the dominated window. In the above example, for document 1, the 3-4 window dominates the 3-5

Algorithm UNORDERED-WINDOWS-NO-DOMINATION

```
1: INPUT: PostingIterator[ ] itrs : One postings list iterator per term.
2: INPUT: width : unordered window parameter.
3: OUTPUT : WindowArray output : Output array of windows
4: while true do
5:     minPos = MINIMUMPOSTION(itrs)
6:     maxPos = MAXIMUMPOSTION(itrs)
7:     if (maxPos − minPos) < width then
8:         output.add(minPos, maxPos);
9:     end if
10:    itr_i.next(), where (p_i = minPos)
11:    if done_i return output;
12: end while
```

window, so, the second window would be omitted. Similarly, in document 5, the 2-8 window would be omitted, as it is dominated by the 2-3 window. This restriction reduces the number of extracted unordered windows in the above example from 6 to 4. The UNORDERED-WINDOWS-NO-DOMINATION algorithm details the process that extracts non-dominated windows. This algorithm only processes each posting list once. So, we can bound the complexity at $\mathcal{O}(\sum_i^n |pos_i|)$. For longer lists of positions, or larger values of n, this algorithm should be considerably more efficient than the previous algorithm.

Finally, it is possible to assert that each position can only be used in one window instance. The UNORDERED-WINDOWS-NO-REUSE algorithm details a process that extracts all windows that do not share any term instances. This restriction reduces the set of unordered window instances from 6 to 3. In document 1, the 3-5 window is dropped, and in document 5, both the 2-8, and the 6-8 windows are dropped. Similar to the UNORDERED-WINDOWS-NO-DOMINATION algorithm, this algorithm only processes each posting list once, so we bound the complexity of this algorithm at $\mathcal{O}(\sum_i^n |pos_i|)$. However, some minor efficiency improvements over the UNORDERED-WINDOWS-NO-DOMINATION may be observed, through a reduction of the number of windows checked by the algorithm.

Algorithm UNORDERED-WINDOWS-NO-REUSE

```
1: INPUT: PostingIterator[ ] itrs : One postings list iterator per term.
2: INPUT: width : unordered window parameter.
3: OUTPUT : WindowArray output : Output array of windows
4: while true do
5:     minPos = MINIMUMPOSTION(itrs)
6:     maxPos = MAXIMUMPOSTION(itrs)
7:     if (maxPos − minPos) < width then
8:         output.add(minPos, maxPos);
9:         for itr_i in itrs do
10:            itrs[i].next()
11:            if itrs[i] is done; return output
12:        end for
13:    else
14:        itr_i.next(), where (p_i = minPos)
15:        if done_i return output;
16:    end if
17: end while
```

The original study proposing these windows (Metzler and Croft, 2005) did not specify exactly how positions can be reused in window instances. The canonical implementation of these window functions, Indri [1], uses the UNORDERED-WINDOWS-NO-DOMINATION algorithm. Similar algorithms, based on the same three assumptions of term instance reuse, are also possible for ordered windows. However, when the width of the ordered window is set at 1, there is no difference in the set of extracted windows between the three algorithms. We empirically investigate the performance of these three algorithms below in Section 5.6.1.

5.4 Monolithic Index Construction

A key requirement of any index data structure is that it can be constructed in a scalable manner, as discussed in Chapter 3. Many inverted index construction algorithms can be viewed as a variant of a large scale sorting algorithm. It takes as

[1] A component of The Lemur Project, `http://www.lemurproject.org/indri.php`

Algorithm MEMORY-BASED ONE-PASS

```
1: for i ← 1 to M do
2:     for t_j ∈ D_i do
3:         S[t_j][i] += 1
4:     end for
5: end for
6: for t_j ∈ domain(S) do
7:     output term t_j; start new posting-list
8:     for i ∈ domain(S[t_j]) do
9:         append posting (i, S[t_j][i])
10:    end for
11: end for
```

input a sequence of terms or term dependency instances, as observed in documents or other retrieval units, and produces as output a data structure that collects instances of like-terms in posting lists. Index construction algorithms that require at most $O(|C| \log |C|)$ execution time are considered scalable. It is important to note that the length of the collection should be measured in the units being indexed, not necessarily in terms.

In this section, we present previously studied indexing algorithms. Each algorithm has already been shown to be scalable in a monolithic sense (Witten et al., 1999). These algorithms are presented as a comparison point for later algorithms that construct frequent and sketch indexes.

5.4.1 Inverted Count Indexes

We start with some well-known textbook approaches to the construction of inverted indexes of terms extracted from a collection of textual documents. The input of each algorithm is a collection, C, composed of a set of documents, $\{D_1, \ldots, D_M\}$, to be indexed. Each document can be parsed directly into a sequence of indexable terms, $\{t_0, \ldots, t_{|D_m|}\}$.

The in-memory inversion algorithm, MEMORY-BASED ONE-PASS, operates using S, a dictionary data structure that is indexed by terms, with $S[t]$ storing the set of

Algorithm Disk-Based One-Pass

```
1: for i ← 1 to M do
2:     for t_j ∈ D_i do
3:         append (t_j, (i, 1)) to the output file F
4:     end for
5: end for
6: sort F, coalescing entries (t_j, (i, tf_1)) and (t_j, (i, tf_2)) to (t_j, (i, (f_1 + f_2))) as
   soon as they are identified
7: for (t_j, ℓ) ∈ F do
8:     output term t_j; start new posting-list
9:     for (i, tf_{t_j,i}) ∈ ℓ do
10:        append posting (i, tf_{t_j,i})
11:    end for
12: end for
```

document frequencies in S at which the term t_j appears. In practice, S is a hash table or a search tree of some sort, each element of which is a linked list or variably-sized array. It can clearly be seen that this algorithm, for even relatively small collections, will require significant amounts of main memory. In the worst case, every term in the collection is unique. The memory requirement is, therefore, only bounded by the size of the input data, $|C|$. For this reason, we assert that this algorithm is not a scalable index construction algorithm.

The first (and widely-known) improvement to the simple mechanism is to use a post-processing sorting phase to bring together the occurrences of each term, rather than require a dynamic data structure (Witten et al., 1999). In this process, described as algorithm Disk-Based One-Pass, every possible term in the collection is added to an output set F, stored on disk as a sequential file. That file is then sorted into term order, and like-terms are collected together to form the index. Sorting is a well understood problem, even when only sequential-access storage is available, and in this approach, the per-item cost of sorting approximately corresponds to the cost of constructing S in algorithm Memory-Based One-Pass, depending on exactly which data structure is used to represent S.

The drawback of this Disk-Based One-Pass approach is that F contains one record for each of the terms present in the collection, C, meaning that the peak disk space required can be as big as C. Moreover, this amount of space is required regardless of the number of repeated terms. On the other hand, random-access data structures are not required – the space required by the dictionary S of the Memory-Based One-Pass algorithm is moved to disk, and processed sequentially via the sorting algorithm, rather than as an in-memory search structure.

Further improvements can be made through the use of an in-memory buffer, and a cascading, coalescing step. The step is undertaken every time an in-memory output buffer of moderate size is filled with terms. The peak disk space requirement might be reduced, because frequently-occurring terms will be stored many fewer times.

5.4.2 Indexes of Term Dependencies

5.4.2.1 Full Index

The only required modification to the Disk-Based One-Pass algorithm in order to create full indexes of term dependencies, is the extraction of term dependencies, instead of terms, in step 2.

Clearly, extracting different types of windows will change the intermediate and final space requirements. First, the number of records stored in F depends on the type of dependency being extracted. For example, $(100 - 7) \cdot 7 + \sum_{i=1}^{6} i$ instances of `uw-w8-n2` can be extracted from a document of 100 terms. More generically, $(|D| - (w-1))\binom{w-1}{n-1} + \sum_{(n-1) \leq i < (w-1)} \binom{i}{n-1}$ unordered window instances, of width w, containing n terms, can be extracted from a document containing $|D|$ terms.

Second, the space required to store each posting, $(td, (i, tf_{td,i}))$, directly depends on the number of terms in the type of window. Consider extracting ordered windows of width 1, containing n terms, also called n-grams. As each window is extracted from

Algorithm Disk-Based Positional One-Pass

```
1: for i ← 1 to M do
2:     for t_j ∈ D_i do
3:         append (t_j, (i, j)) to the output file F
4:     end for
5: end for
6: sort F, coalescing entries (t_j, ℓ_1) and (t_j, ℓ_2) to (t_j, ℓ_1 ∪ ℓ_2) as soon as they are
   identified
7: for (t_j, ℓ) ∈ F do
8:     output term t_j, and posting list ℓ
9: end for
```

the document, each term will occur in at least n windows. So, the space required to store the set of all extracted windows could reach $n \cdot |C|$.

5.4.2.2 Positional Index

To construct a positional index, the Disk-Based One-Pass algorithm must be modified to extract and collect sets of document positions, rather than document frequency values. Disk-Based Positional One-Pass details the modification.

5.5 Distributed Index Construction

Recall from Chapter 3 that in a parallel computing environment, scalability has a slightly different meaning. Now, we required that, as the problem size is increased and the number of processors is increased by a matching proportion, a distributed implementation of the algorithm executes in time that remains fixed, or grows only slowly. If a problem of size $|C|$ can be executed on a single processor in time t, then a problem of size $p \cdot |C|$ when distributed over p machines, should be solvable in time not significantly greater than t.

It is important to note that the desired output will influence the scalability of the distributed indexing algorithm. The final index files can be output as a set of shards across a subset of processing nodes. Alternatively, it could be combined into a

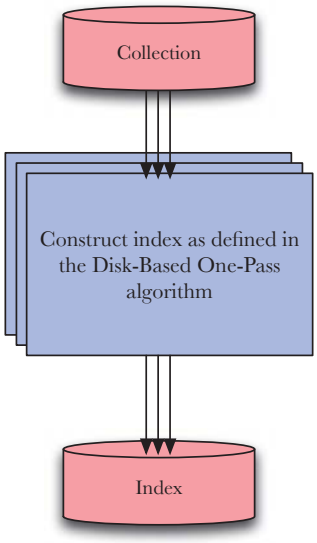


Figure 5.2: Example parallel processing diagram showing how inverted indexes of term dependencies can be constructed in parallel.

monolithic file on one processor. A monolithic index is likely to be more efficient for retrieval algorithms, as postings data does not need to be merged at retrieval time. In this thesis, we assume that the aim is to construct one shard per processing node and any merging of index shards is conducted as a separate process.

Given this assumption, the simplest approach to distributed indexing is to assign each processor a fraction of the input documents and use one of the above monolithic indexing algorithm on each processing node. This algorithm produces one index shard per processor.

In some circumstances, this approach is not feasible, particularly when global collection statistics are required to filter or discard some vocabulary entries. In these cases, information must be passed between processing nodes during indexing. We further discuss parallel indexing algorithms in Chapter 6.

5.6 Experiments

5.6.1 Identifying Windows using Positional Data

As discussed earlier, several different algorithms are possible for the combination of positional data to form ordered and unordered windows. These algorithms stem from three different possible assumptions about the reuse of terms in successive window instances. Importantly, these decisions may have some impact on the efficiency of window extraction, and on the effectiveness of the extracted windows for the retrieval model. In this section, we investigated algorithms that span the three algorithms presented above in Section 5.3.2.

The Unordered-Windows-No-Reuse algorithm asserts that if a term is used in a window instance, then it can not be used in any other instances. This is the strictest assumption of term reuse in windows. The Unordered-Windows-No-Domination algorithm asserts that dominated windows are ignored. A window, w_1, is dominated by another window, w_2, where both windows start at the same location, but w_1 is longer or equal to w_2. Each window must, therefore, have a unique starting location, but may share other terms with other window instances. Finally, the Unordered-Windows-All algorithm allows all terms to be reused, each unique set of term positions that conform to the width requirements, forms a new window instance.

The canonical implementation of the sequential dependence model, in Indri,[2], uses the Unordered-Windows-No-Domination algorithm. We seek to measure the efficiency/effectiveness trade-off implicit in this choice.

We test the differences between these algorithms in three ways: first, by comparing the collection frequency of a sample of queried windows; second, by comparing retrieval efficiency of each algorithm; and finally, by comparing the retrieval effective-

[2]A component of The Lemur Project, `http://www.lemurproject.org/indri.php`

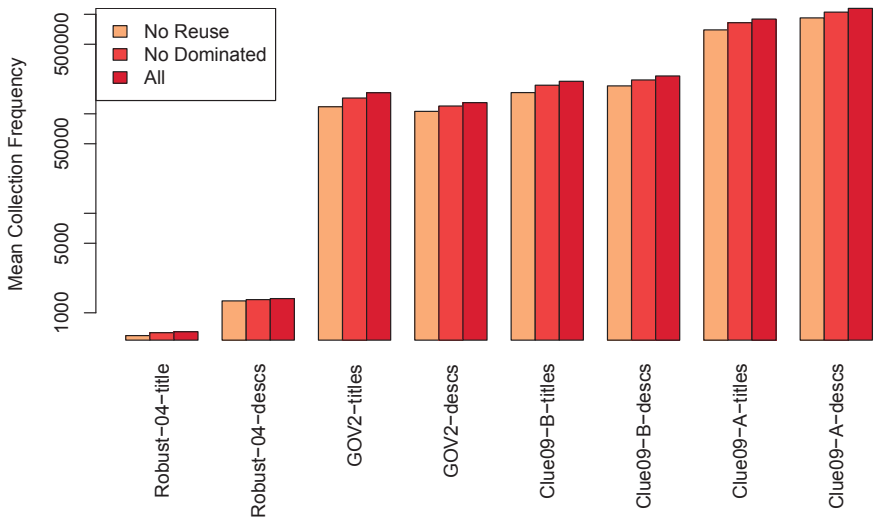


Figure 5.3: Mean collection frequency for all `uw-w8-n2` extracted from all TREC topics, for each tested collection, as extracted by each of the unordered window extraction algorithms.

ness of each algorithm. We compare each of these algorithms in the context of the sequential dependence model (Metzler and Croft, 2005). In this setting, each algorithm for the extraction of ordered windows will return identical results; the terms used to compose ordered windows of width 1 have little potential for reuse. However, unordered windows are affected, therefore we focus our analysis on this type of window.

For each of these experiments, we use 4 TREC collections: Robust-04, GOV2, Clueweb-09-Cat-B and Clueweb-09-Cat-A. A total of 500 topics are available for these collections, as discussed in Chapter 3. We focus on the unordered window features, as used in the sequential dependence model (SDM) for each of these experiments.

Figure 5.3 shows the mean collection frequency of unordered windows using each of the three algorithms, for each TREC collection. As expected, this data shows that the Unordered-Windows-No-Reuse algorithm produces fewer window instances

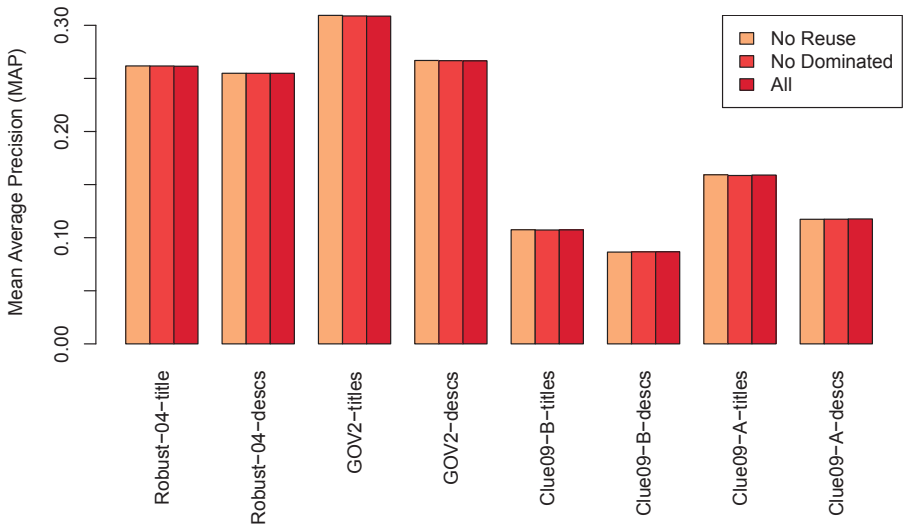


Figure 5.4: Mean average precision for the sequential dependence model, for all TREC topics, for each tested collection, using each of the window extraction algorithms.

than the Unordered-Windows-No-Domination algorithm, and both produce fewer window instances that the Unordered-Windows-All algorithm.

The differences between the three algorithms is relatively small. The frequencies of `no-reuse` extracted windows are around 10% lower than the `no-domination` extracted windows, and the frequencies of `all` windows are around 7% lower than `no-domination` extracted windows, across all collections and both types of queries.

Next, we investigate how retrieval effectiveness is affected by each algorithm. Figure 5.4 shows how each algorithm effects retrieval effectiveness, as measured using mean average precision. We can clearly see that there is almost no change in effectiveness for each window extraction algorithm.

We observe similar results for other retrieval metrics: nDCG@20, P@20, and ERR@20. For each metric, the Fisher randomization test was applied to all pairs of results, for each collection and query type, no significant differences were observed ($\alpha = 0.05$).

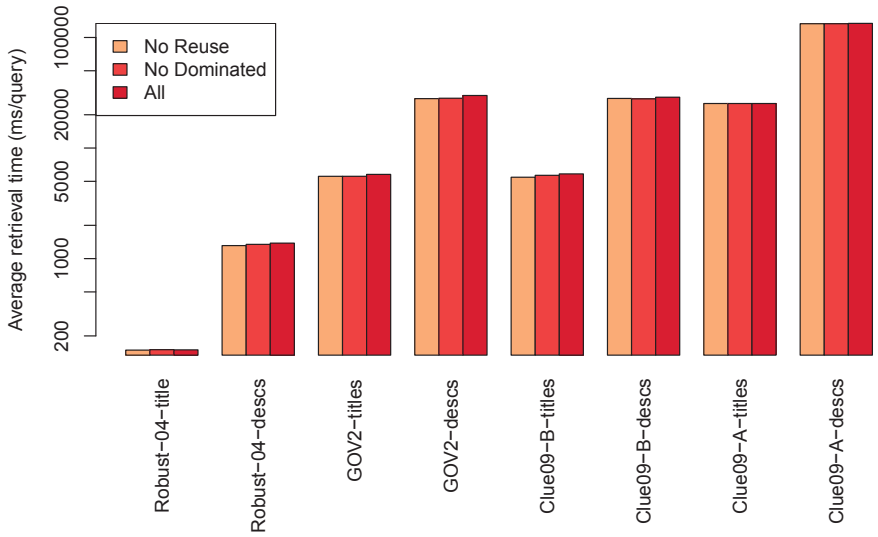


Figure 5.5: Time to return the top 1000 documents, using SDM, for each collection, using each window extraction algorithm. Reported times are the mean per-query time of 5 repeated executions of all queries.

Finally, we measure retrieval efficiency of each algorithm. Given the complexity of each algorithm, we expect to observe that the `no-reuse` algorithms will execute faster than the `no-domination` algorithms, which will execute faster than the `all` algorithms. Even though the output of the ordered window extraction algorithm remains identical in all cases, both unordered and ordered window extraction algorithms are modified according to the above assumptions. This measure will exaggerate any observed time differences between the assumptions. Figure 5.5 shows the retrieval efficiency of each algorithm. The retrieval efficiency of each algorithm is measured as the time to retrieve the top 1000 documents, averaged over 5 repeated runs of all queries, for each tested collection.

We observe almost no difference in efficiency between the `no-reuse` and `no-domination` algorithms. A small difference is observed between the `no-domination` and `all` algorithms. The standard deviation of the reported mean processing times is computed to be less than 2% of the reported mean for each collection.

From this data, there is no clear reason to prefer any of these window algorithms. Intuitively, the `no-reuse` and `no-domination` algorithms will omit some windows that are nearby in the text. This could make it more difficult to identify useful regions of the document. However, with a relatively small window size, 8, we have observed that this is not a large problem for the available queries.

5.6.2 Distribution of Term Dependencies in English Collections

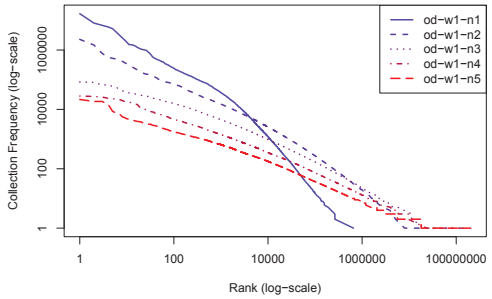
In this section we investigate the distribution and growth rates of ordered and unordered windows in three English TREC collections: Robust-04, GOV2, and Clueweb-09-Cat-B. Each collection is processed using the Krovetz stemmer (Krovetz, 1993). In our simulation of the extraction of window instances, we assume that all positions can be reused. This is equivalent to using using the `all` window recombination algorithms.

This information helps us to estimate the number of entries in full inverted indexes of windows. It also helps us to estimate the distribution of instances across posting lists. Given the potential for very large space requirements, it is vitally important that we can estimate the vocabulary size and posting list lengths of full indexes of term dependencies prior to construction.

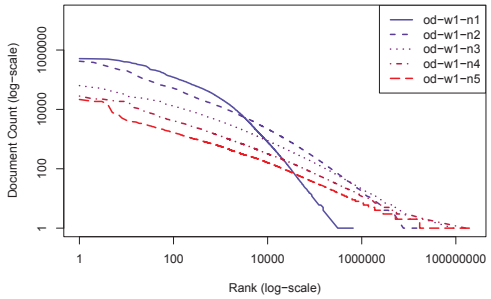
5.6.2.1 Skew

First, we look at the distributions of different types of ordered and unordered windows. This data will allow us to directly estimate the space required to store the vocabulary and posting list data for full indexes of window data for each collection. Skew in this context refers to the relationship between the frequency or document count of a term and the rank of the term (when ranked by the appropriate collection statistic).

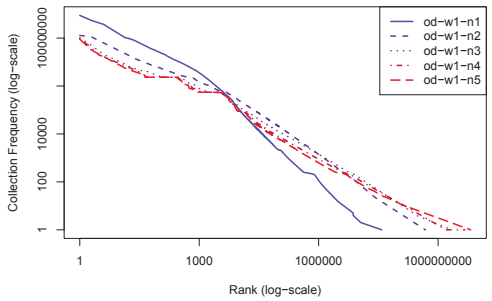
The distribution of ordered windows of width 1, containing 1 to 5 terms, (`odw-w1-nN` for $1 \leq N \leq 5$), as measured using the collection frequency and document count



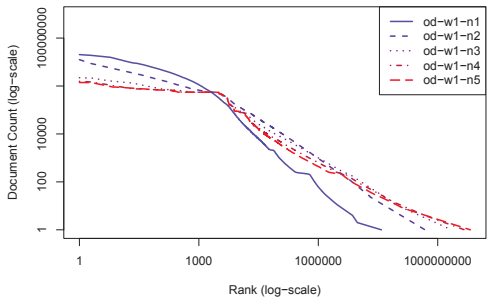
(a) Robust-04, Collection Frequency



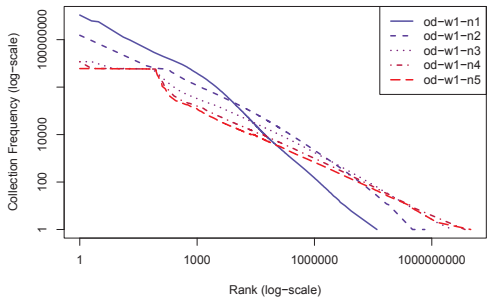
(b) Robust-04, Document Count



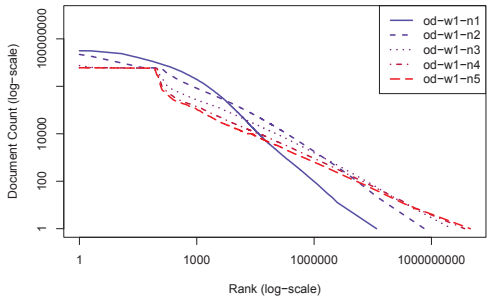
(c) GOV2, Collection Frequency



(d) GOV2, Document Count



(e) Clueweb-09-Cat-B, Collection Frequency



(f) Clueweb-09-Cat-B, Document Count

Figure 5.6: Skew in the distribution of ordered windows of width 1, with 1 to 5 terms, for the Robust-04, GOV2, and Clueweb-09-Cat-B collections. Graphs (a), (c) and (e) shows skew in collection frequency, Graphs (b), (d) and (f) shows skew in document count, for each ordered window. Note that ordered windows of width 1, containing n terms are frequently labeled n-grams.

statistics, is shown in Figure 5.6. All of the graphs shown in this figure are Zipfian graphs, relating the collection statistic to the rank of the ordered window, when sorted in decreasing order by the collection statistic. The x and y axes for each collection is held static to enable a direct comparison between collection frequency and document count statistics. The distribution of ordered window instances graphed here can be equivalent to phrases, shingles or n-grams, depending on the precise definition of each of those terms.

This data shows that the distribution of ordered windows of between 1 and 5 terms approximately follows a power-law distribution for each collection. We observe that very few unique windows occur frequently in the collection, and a significant number of unique windows occur very few times in the collection.

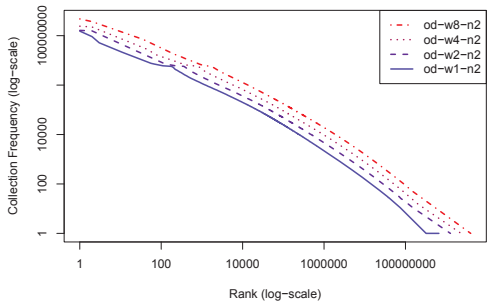
We can also see from this figure that the skew of the vocabulary decreases as the number of terms, N, increases. As the length of the phrase increases, there are fewer frequent windows, and the vocabulary rapidly grows. Interestingly, the the skew in the document count statistics only appears to vary from the skew in the collection frequency statistics for the most frequent items. These very frequent items occur in almost every document in the collection. That is, the observed plateau approaches the maximum value for the document count statistic.

Next, we investigate how the width of windows affects observed skew in each statistic. Figures 5.7 and 5.8 show the skew of extracted ordered and unordered windows, for a range of window widths, all containing two terms, over the collection frequency and document count statistics. We only display data collected from Clueweb-09-Cat-B. Other collections were also tested, and they show similar trends.

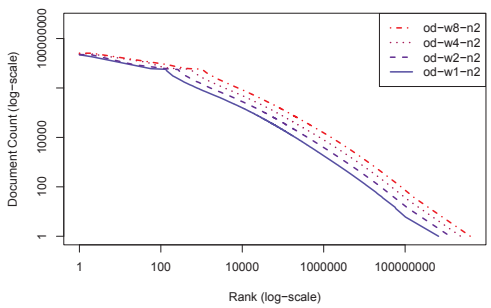
We again observe that both ordered and unordered windows of pairs of terms, across a range of window widths approximately follow power law distributions. As expected, we observe that as the window width increases, the total number of extracted window instances also increases. This increase is exhibited by an increase in

Table 5.1: Top 10 most frequent terms, and windows in three collections.

Rank	od-w1-n1	od-w1-n2	od-w1-n3	uw-w8-n2
Robust-04				
1	the	of the	the united states	of the
2	of	in the	one of the	the the
3	to	to the	the end of	the to
4	and	on the	per cent of	in the
5	in	for the	as well as	and the
6	a	and the	article type bfn	a the
7	that	that the	member of the	and of
8	for	by the	part of the	that the
9	is	with the	in accordance with	for the
10	on	at the	in order to	is the
GOV2				
1	the	of the	image image image	image image
2	of	image image	0 0 0	of the
3	and	in the	img img img	0 0
4	to	0 0	17 jul 2002	the the
5	a	jul 2002	0e 0 00	and the
6	in	to the	image landsat tm	the to
7	for	for the	00 00 00	in the
8	1	by the	0 00 0e	and of
9	image	on the	00 0e 0	00 00
10	0	c2 rdif	18 jul 2002	a the
Clueweb-09-Cat-B				
1	the	of the	the free encyclopedia	of the
2	of	in the	wikipedia the free	the the
3	and	to the	all rights reserved	and the
4	to	on the	one of the	the to
5	a	and the	the terms of	in the
6	in	is a	register trademark of	and of
7	for	for the	privacy policy about	a the
8	is	at the	this page was	is the
9	on	from the	jump to navigation	a of
10	by	with the	retrieve from http	and to

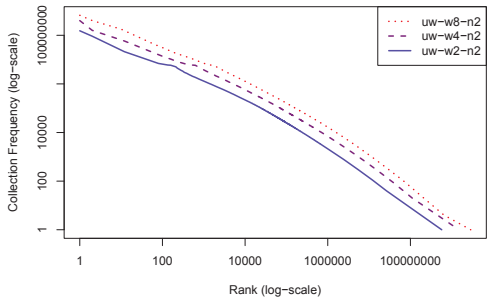


(a) Clueweb-09-Cat-B, Collection Frequency

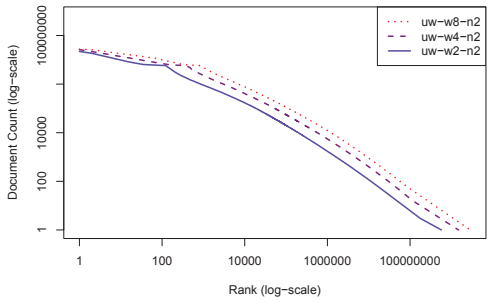


(b) Clueweb-09-Cat-B, Document Count

Figure 5.7: Skew in the distribution of ordered windows of two terms, for varying window widths, for the Clueweb-09-Cat-B collection. Graph (a) shows skew in collection frequency, the graph (b) shows skew in document count, for each ordered window.



(a) Clueweb-09-Cat-B, Collection Frequency



(b) Clueweb-09-Cat-B, Document Count

Figure 5.8: Skew in the distribution of unordered windows of two terms, for varying window widths, for the Clueweb-09-Cat-B collection. The left graph shows skew in collection frequency, the right graph shows skew in document count, for each unordered window.

the area under each curve. Perhaps surprisingly, the skew of this distribution does not appear to change as the width of each type of window is varied. This appears to suggest that the majority of extracted windows simply become more frequent, as the size of the window is increased. Similar to the data shown in Figure 5.6, except for

the most frequent items, the skew over the document count statistic is very similar to the skew over collection frequency statistic.

The anomalous horizontal sections from the Zipfian graphs, (Figures 5.6, 5.7 and 5.8), for the GOV2 and Clueweb-09-Cat-B collections, were investigated. Both collections contain a large degree of frequently replicated header and footer text. The windows extracted from this replicated data cause horizontal sections in the skew graphs. The majority of the replicated data in the Clueweb-09-Cat-B collection originates from the header, sidebar, and footer of the Wikipedia documents included in the collection. This is clearly seen in Table 5.1, which shows the most frequent 10 terms in each collection, for four different types of window.

Finally, we can use this data to estimate the space requirements of a full index of each type of window. Note that these estimates require some assumptions of the average space requirements of each unique window, and each posting list item. We can estimate the space requirements of the Vocabulary, and Posting-List components of a full index of windows as:

$$Vocab. = |V| \cdot n \cdot |term|$$

$$Posting\text{-}List = |V| \cdot |header| + |P| \cdot |posting|$$

Where $|V|$ is the size of the vocabulary, $|term|$ is the average size of a term in bytes, $|header|$ is the average size of the header in bytes, $|P|$ is the number of postings in the index, and $|post|$ is the average size of a posting in bytes. The posting list header is used to store collection level statistics for the term, such as, collection frequency, and document count.

The above analysis allows us to estimate the size of the vocabulary and the number of postings in the index. For example, the average size of a posting will vary depending on the average length of the posting lists. Short posting lists generally contain large delta-encoded document identifiers. It will also affect the header size. Shorter posting

lists will have smaller collection frequency, and document count values, potentially reducing the size of the posting list header. Similarly, the number of documents in the collection will also affect the average size of a posting, since additional documents means that the average delta-encoded document identifier will be larger.

These estimates are not unreasonable, they assume the use of delta-encoding, and the vbyte integer compression. The header stores three integers; the collection frequency, the document count, and the length of the posting list in bytes. Each posting stores two integers, a document count and a document frequency value. However, these are crude estimates, in that the total number of documents in the collection, and the average posting list length, will directly influence the average space required to store each delta-encoded posting (Arroyuelo et al., 2013).

Table 5.2 shows estimated and actual space usage for the three index structures that are required to directly store the features used by the sequential dependence model, for three TREC collections. Here, we assume that each unique window consisting of n words requires $n * 8$ bytes of storage, each posting list has a header of 12 bytes, and each posting requires an average of 2.5 bytes. For simplicity, we do not vary these estimated byte costs for each window type, or collection length.

We can see that the estimated space requirements for these indexes are reasonably close to the actual space requirements. Importantly, we can see that it is feasible to construct even full indexes of these features. Even so, we note that the space requirements for indexes of unordered windows containing two terms is significantly larger than the space requirements of the other indexes. However, the space required by each index decreases, as the size of the collection grows. That is as the collection grows, the space efficiency of full indexes of windows improves. This is, a direct effect of the growth rate of the vocabulary of each type of window.

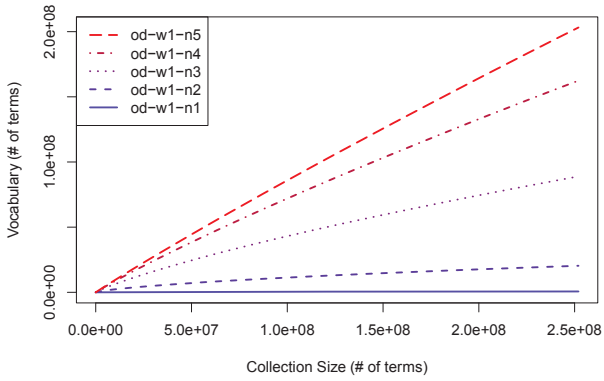
Table 5.2: Estimated and actual space requirements for full indexes that directly store data required to compute each of the SDM features. For comparison purposes, the final column shows the size of the compressed collection, (using GZIP).

Estimated Space Requirements

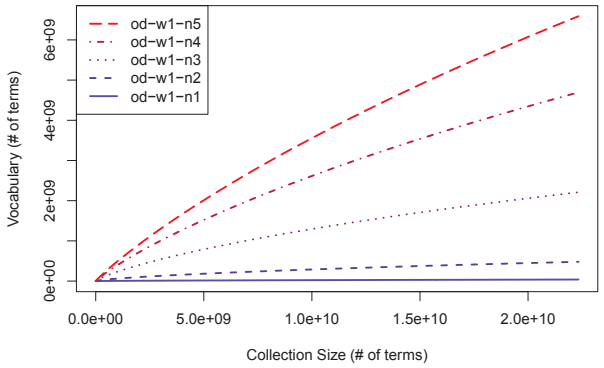
Collection	Vocab.	Posting-List	Combined	Collection
Robust-04				
od-w1-n1	5.01 MB	275 MB	280 MB	583 MB
od-w1-n2	311 MB	741 MB	1051 MB	583 MB
uw-w8-n2	1,325 MB	4,022 MB	5,347 MB	583 MB
GOV2				
od-w1-n1	294 MB	13,987 MB	14,280 MB	80,000 MB
od-w1-n2	7,321 MB	37,087 MB	44,408 MB	80,000 MB
uw-w8-n2	31,384 MB	201,222 MB	232,606 MB	80,000 MB
Clueweb-09-Cat-B				
od-w1-n1	301 MB	25,281 MB	25,582 MB	145,000 MB
od-w1-n2	10,095 MB	56,915 MB	67,010 MB	145,000 MB
uw-w8-n2	43,835 MB	321,590 MB	365,425 MB	145,000 MB

Actual Space Requirements

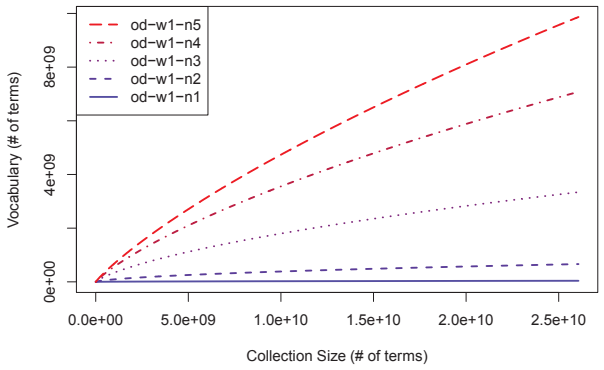
Collection	Vocab.	Posting	Combined	Collection
Robust-04				
od-w1-n1	4.42 MB	240 MB	244 MB	583 MB
od-w1-n2	247 MB	707 MB	955 MB	583 MB
uw-w8-n2	1,103 MB	3,895 MB	4,998 MB	583 MB
GOV2				
od-w1-n1	426 MB	12,660 MB	13,087 MB	80,000 MB
od-w1-n2	6,061 MB	40,393 MB	46,455 MB	80,000 MB
uw-w8-n2	27,779 MB	214,716 MB	242,496 MB	80,000 MB
Clueweb-09-Cat-B				
od-w1-n1	418 MB	22,787 MB	23,205 MB	145,000 MB
od-w1-n2	8,665 MB	64,238 MB	72,903 MB	145,000 MB
uw-w8-n2	39,955 MB	354,360 MB	394,315 MB	145,000 MB



(a) Robust-04

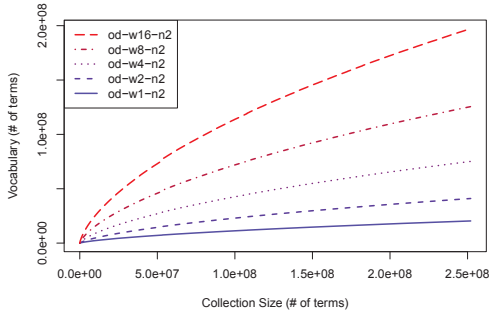


(b) GOV2

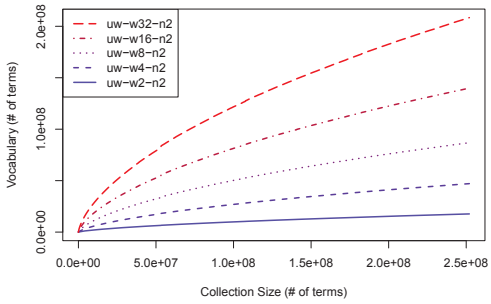


(c) Clueweb-09-Cat-B

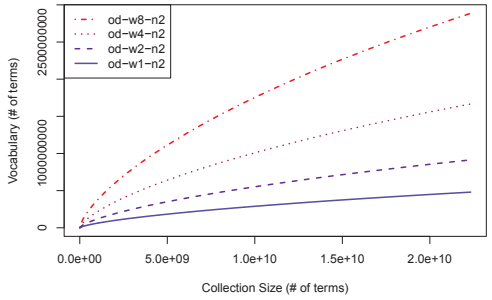
Figure 5.9: Growth rate of the vocabulary of ordered windows of width 1, with varying numbers of terms, for each collection. Note that ordered windows of width 1 are equivalent to n-grams.



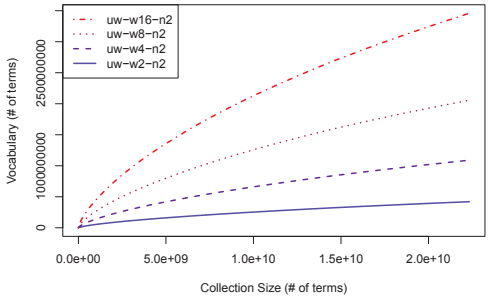
(a) Robust-04, od-wW-n2



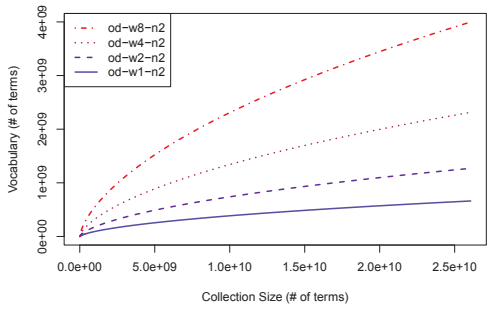
(b) Robust-04, uw-wW-n2



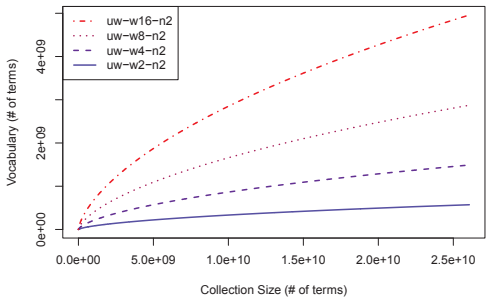
(c) GOV2, od-wW-n2



(d) GOV2, uw-wW-n2



(c) Clueweb-09-Cat-B, od-wW-n2



(d) Clueweb-09-Cat-B, uw-wW-n2

Figure 5.10: Growth rate of the vocabulary of ordered and unordered windows containing two terms, for a range of window widths, for the Robust-04, GOV2, and Clueweb-09-Cat-B collections.

5.6.2.2 Growth Rates

We have observed the relationship between the size of the vocabulary of windows and the space requirements of the index. We now investigate the growth rate of the vocabulary of a variety of different types of ordered and unordered windows. In this section, we measure the relationship between the length of the collection and the total vocabulary size. Data collected here allows us to estimate the approximate change in vocabulary as collection size grows.

In these experiments we again use three TREC Collections: Robust-04, GOV2, and Clueweb-09-Cat-B. In each of these experiments, the order of the documents in the collection is randomized to remove anomalies caused by the order of documents.

Figure 5.9 shows the growth rate of the vocabulary of ordered windows of width 1, (`od-w1-nN`, for $1 \leq N \leq 5$), in each collection. Recall that this type of window is equivalent to phrases, shingles or n-grams, depending on their specific definitions. This type of graph is commonly called a Heaps' law graph.

Even though on these graphs the single term vocabulary appears flat, it is actually growing according to Heaps' 'law'. The scale of each graph makes the growth rate appear flat.

As the number of terms in each ordered window increases, the vocabulary growth increases dramatically. Almost every 5-gram extracted from Robust-04 is unique, the total vocabulary of 5-grams is 80.6% of the collection length. For GOV2 and Clueweb-09-Cat-B, the total vocabulary of 5-grams is 29.5%, and 37.9% of the collection length, respectively. This statistic contrasts with the total vocabulary of single terms, which is less than 1% of the length of each collection. This data shows the scale of the problem of storing the full vocabulary of windows of multiple terms.

Figure 5.10 shows the growth rate of vocabulary of ordered and unordered windows, as the width of the window is increased, for the Robust-04 collection. These graphs are very similar to the graphs showing the effect of increasing the number of

terms in each window. The growth rate of the vocabulary increases with the width of the window.

In this case, the length of the collection and the total vocabulary can no longer be directly compared. Recall that the total number of window instances is larger than the collection size. For unordered windows of width 8, containing 2 terms, the vocabulary is actually 5.0%, 1.3%, and 1.5% of the total unordered window instances extracted from Robust-04, GOV2, and Clueweb-09-Cat-B, respectively. This observation matches observations made for the skew in the distribution of unordered windows. The ratio between the vocabulary, and the total number of instances extracted, does not change dramatically, as the width of the window is increased.

5.7 Summary

In this chapter, we presented existing index structures for term dependency data. We discussed modifications to existing index construction algorithms that enable the efficient construction of each type of index. These index structures form a baseline against which the frequent and sketch index data structures will be compared.

We investigated the application of three assumptions that could allow for a trade-off between retrieval efficiency and effectiveness, in the extraction of window instances from positional posting lists. Empirically, we find that there is no reason to prefer any particular type of extraction algorithm. While the extracted collection statistics varied between the algorithms, the retrieval efficiency and retrieval effectiveness did not noticeably change across the set of algorithms. The canonical implementation of SDM, and WSDM, uses the `no-domination` algorithms. Throughout the rest of this thesis, we follow their example.

We intend to investigate differences between the performance of each of these algorithms for different retrieval model features in future work. Specifically, larger window sizes may dramatically change the effect of each assumption. Additionally,

the choice of window algorithm may have a significant effect on smaller retrieval units, such as passages or sentences, or short documents, such as microblogs. Further research into the different assumptions of window extraction algorithms could also be broadened to include algorithms for detecting spans, and other types of positional term dependencies.

We investigated the skew in vocabulary and the growth rates for various different windows. This data allows us to make predictions about the vocabulary sizes, and space requirements of full indexes of term dependencies for a range of window types. We evaluated these predictions by comparison to actual full indexes.

In future work, it would be informative to investigate distributions of these windows for different languages. It is also important to investigate distributions of these windows for artificially generated data, particularly large samples of randomly generated Zipfian data. This would help inform us about the differences and similarities between a specific language and these randomly generated symbols.

We observe several stopword-like window instances in the top 10 most frequent dependencies (in Table 5.1). For this initial work, we do not omit any stopword-like windows from any indexes. We did consider removing windows that contain only stopwords, as these are unlikely to improve retrieval performance, ("he said", "she said"). However, it is reasonable to expect that some sequences of only stopwords may be semantically meaningful; a common example is "to be or not to be". Future research will involve investigation into the retrieval utility of these windows of stopwords. A simple automatic method of constructing lists of stop-dependencies may be to discard any window instance that contains a high fraction of stopwords.

Finally, different compression techniques for the compression of posting lists in the full dependency index may change the ratio between vocabulary and posting list space requirements. In future work, an investigation of alternative compression

schemes for term dependency indexes may provide higher compression ratios, without compromising decode rates.

CHAPTER 6

FREQUENT INDEX

6.1 Frequent Index Data Structure

An important observation on the distribution of various types of windows in Chapter 5 is that a large fraction of the vocabulary of windows occurs very few times in the collection. Following this observation, it is reasonable to expect that an index only needs to store data for ordered and unordered windows that occur *frequently* in the target collection. In this context, a frequent window is defined with respect to a threshold, h, and the collection frequency of the window, f_w. To be retained in a frequent index of window data, the collection frequency of a window must be greater than or equal to the threshold, $f_w \geq h$.

From data presented in Chapter 5 on the distribution of windows in English, it is reasonable to expect that this type of filter will be able to discard a large fraction of the vocabulary, and each corresponding short posting list, thereby, greatly reducing the space requirements of any inverted index structure.

A frequency based threshold is not the only possible vocabulary filter. We could alternatively filter windows based on the the number of documents that contain the window, the inverse document frequency, or any other term or window weighting function. The algorithms presented in this chapter can be modified to allow these different types of filtering, where the filtering statistic depends on one or more aggregate statistics. However, the study of alternate filtering mechanisms is out of the scope of this chapter.

As discussed in Chapter 2, recent research has investigated some of these alternative filtering mechanisms for BM25-based proximity models (Broschart and Schenkel, 2012). Their work focused on indexes that store "impacts" (approximate scores) for a variant of the BM25 model. These index structures are filtered using two methods. First each posting list is limited to store only the top k postings. Second, items with a low inverse document frequency (idf) in the collection are discarded from the index. Their proposed index stores posting impacts for a specific variant of the BM25 scoring function. They use a variant of the BM25-TP model, proposed by (Rasolofo and Savoy, 2003). As we observed in Chapter 4, this model is outperformed by both SDM and WSDM-Internal.

As we will show, our findings match those presented by Broschart and Schenkel (2012) in several regards; space requirements can be reduced without compromising retrieval effectiveness. However, our results also demonstrate that these types of indexes can be constructed efficiently in terms of both time and space requirements.

One of the major contributions of this chapter, indexing algorithms for frequent indexes of n-grams, has been previously published as a conference paper (Huston et al., 2011). This study focused on efficient algorithms for the construction of frequent indexes in the context of text reuse detection. In this chapter, analysis of this structure will cover both the construction and the use of the structure in an information retrieval system, for both ordered and unordered windows, as would be used in the most effective dependence models studied in Chapter 4 (SDM and WSDM-Int).

The major contributions in this chapter include:

- index construction algorithms for a frequent index structure;
- an empirical study of efficient and scalable construction algorithms;
- an empirical study of the effect of frequent indexes on information retrieval effectiveness, using both query log analysis, and annotated TREC query sets; and

Algorithm DISK-BASED FREQUENT WINDOW ONE-PASS

```
1: for i ← 1 to M do
2:     for td_j ∈ D_i do
3:         append (td_j, i, 1) to the output file F
4:     end for
5: end for
6: sort F, coalescing entries (td_j, i, f^1_{i,td_j}) and (td_j, i, f^2_{i,td_j}) to (td_j, i, f^1_{i,td_j} + f^2_{i,td_j}) as
   soon as they are identified
7: for (td_j, i, f_{i,td_j}) ∈ F do
8:     if Σ_i f_{i,td_j} ≥ h then
9:         output (td_j, i, f_{i,td_j})
10:    end if
11: end for
```

- an empirical study of the relationship between the threshold parameter, and the space requirements of the frequent index.

6.2 Index Construction Algorithms

6.2.1 Monolithic Construction Algorithms

6.2.1.1 One Pass Algorithms

We start by introducing modifications to some of the indexing algorithms presented in Chapter 5, to construct frequent index structures. The DISK-BASED WINDOW ONE-PASS algorithm is modified to include a check against the threshold value, h, before writing each posting list.

This algorithm is sketched in DISK-BASED FREQUENT WINDOW ONE-PASS. The input of this algorithm is a collection, C, composed of a set of documents, $\{D_1, \ldots, D_M\}$, to be indexed. Each document can be parsed directly into a sequence of indexable term dependency instances, $\{td_0, \ldots, td_{|D_m|}\}$. Note that the exact length of the document, $|D_m|$, will depend on the type of term dependency extracted.

All term dependencies of the specified type in the collection C are added to an output set F, stored on disk as a sequential file. That file is then sorted into lexicographic order, and frequent dependencies are collected together to form the frequent

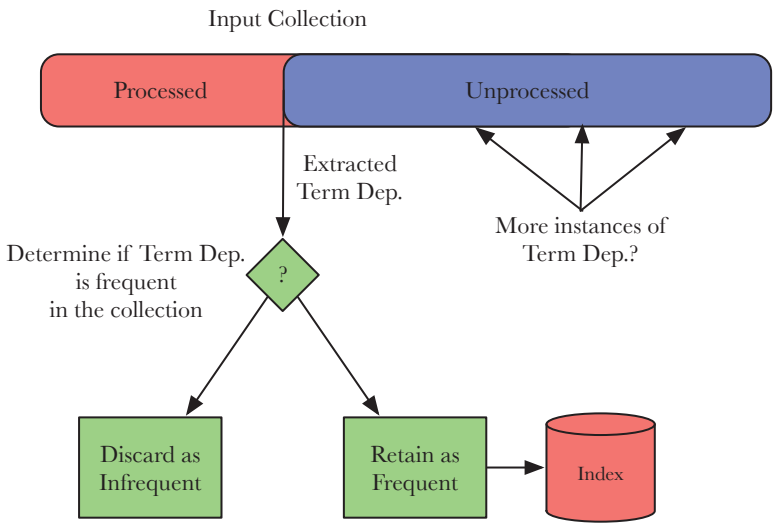


Figure 6.1: The frequency of the extracted term dependency in the collection can not be determined until the entire collection has been processed. This a depiction of the fundamental cause of the space requirements of any one-pass algorithm used to build a frequent index.

index structure. Sorting is a well understood problem, even when only sequential-access storage is available. The time complexity of this algorithm is $\mathcal{O}(|C| \log |C|)$, where the length of the collection is measured in term dependencies.

The drawback of this Disk-Based Frequent Window One-Pass approach is that F contains one record for each of the window instances present in the collection C. For `od-w1-nN`, or n-grams, this means that the peak disk space required is as much as $n+1$ times as big as $|C|$, assuming that a frequency value requires approximately the space of one term. If merge-based sorting is used, and a cascading coalescing step undertaken every time an in-memory output buffer of moderate size is filled with term dependencies, the peak disk space requirement might be reduced, as frequently-occurring dependencies will be stored only once.

The central problem causing the scalability issues of the one-pass algorithms is that it is not possible to predict the true collection frequency of any particular term dependency until *all* documents have been parsed. To ensure the validity of the index,

it is impossible to discard any term dependency instance until the entire collection is processed. This means that the intermediate space requirements of the frequent index structure can be as high, or higher than, the full index structure. Figure 6.1 is a diagram showing the key problem causing the space requirements of the one pass algorithm.

We consider a possible solution; multiple passes over the collection. The initial pass or passes over the collection collect approximate frequency information about all, or a subset of term dependencies. The final pass over the collection checks each extracted term dependency against the collected information. If the term dependency can be identified as *infrequent*, then it can be discarded immediately.

This approach will only be space efficient if space requirements of the intermediate approximate frequency information is significantly smaller that would be required by the Disk-Based Frequent Window One-Pass algorithm. Indeed, a key aim is to limit space requirements to not significantly more than the space required for the final frequent index structure.

6.2.1.2 Hash-based Two-Pass Algorithms

The next indexing algorithm, Hash-Based Window Two-Pass, builds on two independent observations; first, that hash-derived surrogates can be stored instead of the term dependency, saving space; and second, that the actual locations of the dependencies need only be collected once the values of the repeated term dependencies have been identified. The key idea is that the probabilistic hash filter S certainly contains the hash-surrogate of every term dependency that *does* occur more than h times in S, and *might* contain the hash-surrogate of some term dependencies that do not.

The first **for** loop at steps 1 to 5 processes the collection and generates a hash value for each extracted term dependency. A hash value for each dependency is computed

Algorithm HASH-BASED WINDOW TWO-PASS

```
1: for i ← 1 to M do
2:     for td_j ∈ D_i do
3:         compute surrogate hash value; s ← hash_b(td_j)
4:         append (s, 1) to the output file F_1
5:     end for
6: end for
7: sort F_1, coalescing paired entries (s, f_1) and (s, f_2) to (s, f_1 + f_2) as soon as they
   are identified
8: for (s, f) ∈ F_1 do
9:     if f ≥ h then
10:        append s to the output file F_2
11:    end if
12: end for
13: S ← read(F_2)
14: for i ← 1 to M do
15:    for td_j ∈ D_i do
16:        s ← hash_b(td_j)
17:        if s ∈ S then
18:            append (td_j, i, 1) to the output file F_3
19:        end if
20:    end for
21: end for
22: apply steps 6 to 10 of DISK-BASED FREQUENT WINDOW ONE-PASS to the file
    F_3
```

via the function $hash_b(td_j)$, and recorded to an output file F_1, together with an initial frequency count of one. It is assumed that a b-bit hash-surrogate, s, is generated, and that each term dependency, td_j, is thus condensed into a non-unique b-bit string on a many-to-one basis. File F_1 is then processed to generate a set of hash values that correspond to term dependencies that might occur more than m times in S (steps 7 to 12), and that set of hash values is used (step 17) to (perhaps greatly) reduce the number of full term dependency that get processed by step 22.

In terms of disk space used, file F_1 still contains as many as $|C|$ records, even if the sorting and coalescing process indicated at step 6 is carried out in a cascading, block-interleaved manner. But now each record is a b-bit integer plus a frequency

count (the latter requiring only $\lceil \log_2(h+1) \rceil$ bits), and for $b = 40$ and it would be reasonable to bound the threshold, $h \leq 255$. The requirement for file F_1 is then, at most $5 \cdot |C|$ bytes. File F_2 is never larger than F_1, and so does not affect the peak disk space requirement.

The third disk file, F_3, is potentially very large – for pathological sequences, as large as the approximately $(n+1) \cdot |C|$ *terms* required by algorithm Disk-Based Frequent Window One-Pass. But note that it only contains information about term dependencies that either (a) do indeed appear in C more than h times; or (b) by chance, hash to a value that corresponds to some other term dependency(s) that collectively appear more than h times in C. That is, provided that the number of false positives is controlled, the size of F_3 is primarily determined by the total number of occurrences in C of term dependencies that appear more than h times, and (disregarding any compression that might be applied to the final index) is of the same magnitude as the frequent index of term dependencies that is being generated.

In an implementation, the set S can be stored as a bit-vector of 2^b bits for $O(1)$-time random access (which is, of course, impractical when b is larger than around 32 or so); or can be stored as a sorted array of b-bit integers with a logarithmic access cost; or can be stored using a compressed queryable structure that requires less space than either of these two alternatives, while still providing logarithmic-time lookup. Sanders and Transier (2007) describe one such hybrid mechanism.

To gauge the extent to which these various costs become bottlenecks, consider a scenario in which a collection of 1 billion words ($|C| = 10^9$ words, corresponding to around 10–20 GB of HTML data) is being processed as n-grams, where $n = 5$, and $h = 2$. Further, suppose that on average one n-gram in two in the file occurs more than h times, and that the average number of repetitions for n-grams that do reoccur is 4. If these estimates are appropriate, then there is a vocabulary of around 115×10^6 distinct n-gram that pass the threshold. Under these assumptions, and with $b = 40$,

Table 6.1: Percentages of n-grams in three categories: "single", distinct n-grams appearing exactly once; "multi", distinct n-grams appearing more than once; and "repeat", the total of the second and subsequent appearance counts of the n-grams that appear more than once. In total, each row in each section of the table adds up to 100%, since every one of the N n-grams in each file is assigned to exactly one of the three categories.

n	$\|C\| = 250 \times 10^6$			$\|C\| = 500 \times 10^6$		
	single	multi	repeat	single	multi	repeat
1	0.1	0.2	99.7	0.1	0.1	99.7
2	5.5	3.7	90.7	4.6	3.1	92.3
3	27.3	9.2	63.4	24.5	8.7	66.8
4	52.9	10.0	37.1	50.6	10.2	39.2
5	68.4	8.4	23.2	67.8	8.7	23.6

(a) TREC Newswire data (Disks 1-5 of TREC)

n	$\|C\| = 250 \times 10^6$			$\|C\| = 1{,}000 \times 10^6$		
	single	multi	repeat	single	multi	repeat
1	0.3	0.3	99.4	0.2	0.2	99.6
2	6.5	4.3	89.2	4.0	3.0	92.9
3	26.3	9.6	64.0	19.0	8.6	72.4
4	46.7	10.7	42.6	37.4	11.3	51.3
5	58.8	9.8	31.4	49.9	11.4	38.7

(b) TREC GOV2 data

n	$\|C\| = 250 \times 10^6$			$\|C\| = 1{,}000 \times 10^6$		
	single	multi	repeat	single	multi	repeat
1	0.5	0.4	99.0	0.4	0.3	99.3
2	8.3	4.9	86.8	6.1	3.8	90.1
3	26.2	9.5	64.3	21.9	8.5	69.6
4	40.9	10.4	48.7	37.1	10.2	52.7
5	48.4	9.9	41.6	46.1	10.0	43.9

(c) TREC ClueWeb-B data

file F_1 might be as large as 6 GB; file F_2 contains a little more than 115×10^6 entries (the extras arising because of the probabilistic nature of the filter H) and occupies around 700 MB; and file F_3 contains a little over 5×10^8 entries, each occupying 6 words (24 bytes), for a total of 11 GB. As well, during the second pass, file F_2 must be present in memory as an array-based lookup structure H, meaning that of the order of 700 MB of main memory is required through steps 14 to 20 of HASH-BASED WINDOW TWO-PASS. In terms of disk traffic, several gigabyte-sized input files must be sequentially processed, including being sorted.

The final n-gram index must then include 115×10^6 5-gram descriptions, plus, for each of them, an average of four postings, each requiring 2 bytes, and a posting list header of 24 bytes, for a total uncompressed requirement of approximately 6 GB. Non-trivial savings arise once the list of n-grams is compressed, since they can be stored as incremental differences relative to each other (Witten et al., 1999).

If the sequence is enlarged by a factor of 10 – to process 100–200 GB, of source text, say – the memory space required pushes towards the limits of plausibility. If the factor is instead 100, and 1 TB of HTML data is to be processed, the memory requirement for bit vector S – perhaps 25 GB – makes HASH-BASED WINDOW TWO-PASS intractable in terms of memory space required.

The scenario portrayed in here is clearly within the limits of what might be achieved with a single "standard" computer; and the data listed in Table 6.1. The data presented in this table was extracted from the distributions of ordered windows, presented in Chapter 5.

6.2.1.3 Multi-Pass Algorithms

The SPEX MULTI-PASS approach to finding repeated n-grams was developed by Bernstein and Zobel (2006). It makes $n + 1$ passes through the collection C to construct a set of n probabilistic filters, building on the key observation that any

Algorithm SPEX MULTI-PASS

1: compute S_1, a list of the 1-grams in the collection C, together with their occurrence frequencies
2: allocate an empty hash filter S_2
3: **for** $k \leftarrow 2$ **to** n **do**
4: **for** $i \leftarrow 1$ **to** M **do**
5: **for** $j \leftarrow 1$ **to** $|D_i| - 1$ **do**
6: $td_{j,k-1} \leftarrow j^{th}$ term dependency of size $k-1$
7: $td_{j+1,k-1} \leftarrow j+1^{th}$ term dependency of size $k-1$
8: $s_1 \leftarrow hash_b(td_{j,k-1})$
9: $s_2 \leftarrow hash_b(td_{j+1,k-1})$
10: **if** $S_{k-1}[h_1] \geq m$ **and** $S_{k-1}[h_2] \geq m$ **then**
11: $td_{j,k} \leftarrow j^{th}$ term dependency of size k
12: $s_3 \leftarrow hash_b(td_{j,k})$;
13: **if** $S_k[h] < m$ **then**
14: $S_k[h] \leftarrow S_k[h] + 1$
15: **end if**
16: **end if**
17: **end for**
18: **end for**
19: free S_{k-1}, and reallocate a new hash filter S_{k+1}
20: **end for**
21: **for** $i \leftarrow 1$ **to** N **do**
22: $td_{j,n} \leftarrow j^{th}$ term dependency of size k
23: $s \leftarrow hash_{b_2}(td_{j,n})$
24: **if** $h \in S_n$ **then**
25: append $(s, i, 1)$ to the output file F_3
26: **end if**
27: **end for**
28: apply steps 5 to 10 of DISK-BASED FREQUENT WINDOW ONE-PASS to the file F_3

n-gram consists of two $(n-1)$-grams; and that if either of those two $(n-1)$-grams occurs fewer than h times, then the n-gram in question must also occur fewer than h times. Hence, starting with a vocabulary of 1-grams, successive passes over the source sequence generate increasingly longer sets of potentially frequent n-grams. Two generations of the hash filters are required to be active at any given time.

We note that it is possible to directly extend this algorithm to other types of many-term dependencies. Analogously to n-grams, ordered and unordered windows

that contain n terms, can be decomposed into windows that contain $n-1$ terms. For example, an unordered windows of width 12 containing 3 terms can be considered infrequent if an unordered windows of width 12 containing 2 of the same terms is also infrequent. However, for the simplicity of this discussion, we will focus on the original application of this algorithm, indexing of frequent n-grams, (`od-w1-nN`).

Bernstein and Zobel argue that the multiple passes made by SPEX Multi-Pass are justified because the memory space required for the filters is reduced compared to any possible "all in one" approach that seeks to directly generate a hash-filter for n-grams. In fact, compared to using a wide b-based hash in the first pass of Algorithm Hash-Based Window Two-Pass, the additional passes serve little purpose except to save the disk space used by the file F_1 (required in algorithm Hash-Based Window Two-Pass), and come at a considerable cost in terms of execution time. In particular, the SPEX Multi-Pass approach does not result in any saving in terms of memory space, because during the final resolution pass at step 15 when the hash filter S_n is being used, exactly the same amount of memory is required as by algorithm Hash-Based Window Two-Pass for any given level of false positive performance. Nor is the removal of file F_1 a great saving, since file F_1 is smaller than the final file F_3, which is still required if the output of the SPEX Multi-Pass algorithm is to be deterministic rather than probabilistic.

The distributions of terms, measured in Chapter 5, help to explain why the multiple passes of final SPEX Multi-Pass are no more effective than a single pass that generates the n-grams directly. The differences between the distributions of 4-grams and 5-grams is small. That is, as k increases, so too does the probability that any particular repeated $(k-1)$-gram is extended to form a repeated k-gram. Corresponding to this observation, the utility of each filter constructed in SPEX, diminishes. The same effect is also reported in Table 6.1.

Algorithm SPEX Log-Pass

1: **for** $k \leftarrow \{1, 2, 3, 5, 8, \ldots\}$ while $k < n$ **do**
2: apply steps 4 to 15 of SPEX Multi-Pass
3: **end for**
4: apply steps 13 to 21 of Hash-Based Window Two-Pass using the filter H_k

The SPEX Multi-Pass algorithm builds successive hash-filters for $k = 1$, $k = 2$, and so on, through until $k = n - 1$, in the final pass the $n - 1$ filter is applied to the text and the passing n-grams are indexed. In total, n passes through the data are made. The number of passes can be reduced to $\log n$ by pausing at fewer values of k, and checking for more subsequences at step 7. For example, if the values of k are drawn from $\{1, 2, 3, 5, 8, 13, \ldots\}$, then a total of four 5-grams are used in a quest to eliminate each infrequent 8-gram as H_8 is constructed from H_5. We call this approach SPEX Log-Pass, and have tested it as an alternative to the SPEX Multi-Pass approach. It relies on the same assumptions as SPEX Multi-Pass, namely that the number of repeated n-grams for any given value of n is a diminishing fraction of the number of distinct n-grams. As discussed above, these assumptions are not valid for typical text.

6.2.2 Distributed Index Construction

Recall our definition from Chapter 3 that in a parallel computing environment, scalability has a specific meaning. As the problem size is increased and the number of processors is increased by a matching proportion, a distributed implementation of the algorithm executes in time that remains fixed, or grows only slowly. In addition, while there might be a memory-imposed limit on how much data can be loaded on to any single processor, there must not be any overall memory limit imposed. Communication costs must also be bounded as a function of the expected running time, before an algorithm can be argued to be scalable. More precisely, if a problem of size $|C|$ can be executed on a single processor in time t, then a problem of size $p \cdot |C|$

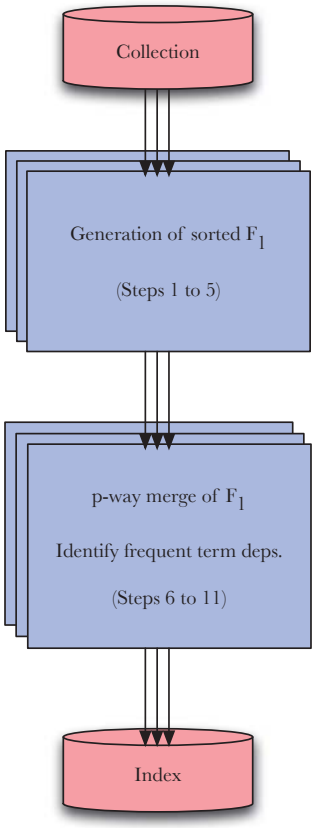


Figure 6.2: Example diagram detailing a distributed implementation of Disk-Based Frequent Window One-Pass. Each replicated box represents a set of p processors. Sets of arrows represent distribution of data across processors.

when distributed over p machines, should be solvable in time not significantly greater than t. Moffat and Zobel (2004) discuss issues to do with performance evaluation in distributed environments.

6.2.2.1 Parallel Indexing

Consider the behavior of the Disk-Based Frequent Window One-Pass, SPEX Multi-Pass, and Hash-Based Window Two-Pass methods if a cluster of computers is available, and the computational load is to be shared across them so as to reduce the elapsed execution time.

Similar to the Disk-Based Frequent Window One-Pass algorithm, discussed in Chapter 5, algorithm Disk-Based Frequent Window One-Pass translates naturally to a distributed processing model. If the input collection C is split into p equal length parts, steps 1–5 can be performed across p processors to make p intermediate files. Those files are then sorted on their respective host machines, before being partitioned into a total of p^2 components that are written to a shared file system so that they can be accessed by all of the machines, and then combined in a set of p independent p-way merge operations, one per machine. This approach to sorting is well understood for parallel computation (see, for example, Tridgell and Brent (1993)) and accounts for steps 6 to 11. Finally, the p separate sorted lists of frequent term dependencies may combined into a single file via another p-way merge operation. Throughout these processing phases all p machines are equally busy, provided that the p^2 subfiles are all of approximately the same size; furthermore, there is little additional work introduced by the partitioning. That is, algorithm Disk-Based Frequent Window One-Pass is scalable in both in a single-processor sense, and also in a distributed sense. However, in a parallel implementation, this algorithm continues to have the drawback of requiring approximately $(n + 1) \cdot |C|$ words of temporary disk space, spread across the p processors. Figure 6.2 shows a high level diagram of a distributed implementation of the Disk-Based Frequent Window One-Pass algorithm.

On the other hand, algorithms Hash-Based Window Two-Pass, SPEX Multi-Pass and SPEX Log-Pass do not readily parallelize. The problem is the amount of memory required by the hash filters S, S_{k-1} and S_k. These arrays must be sized in proportion to the total number of distinct term dependencies that are anticipated to occur in the sequence, and as is shown in Table 6.1, even for curated text such as the Newswire collection, the number of frequent term dependencies is a non-decreasing fraction of the sequence length. Worse, the hash-filters are required, in full, at every

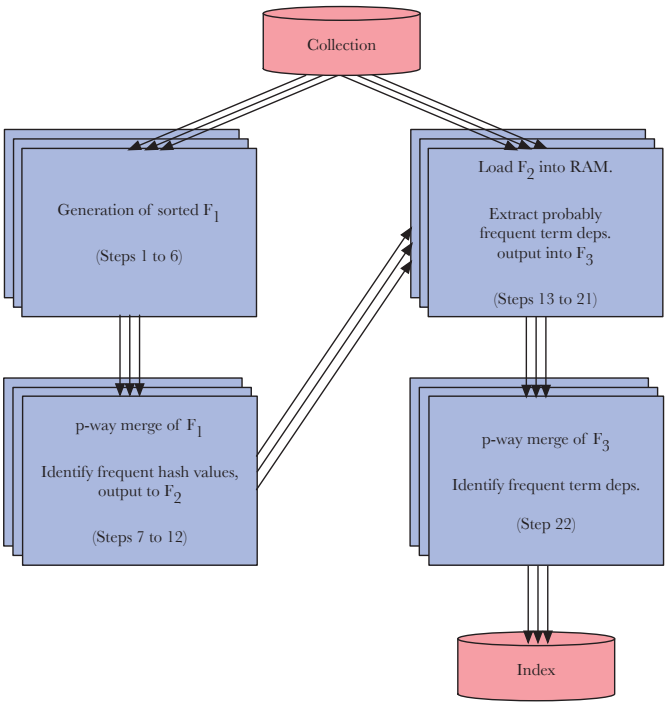


Figure 6.3: Example diagram detailing a distributed implementation of Hash-Based Window Two-Pass. Each replicated box represents a set of p processors. Sets of arrows represent the distribution of data across processors.

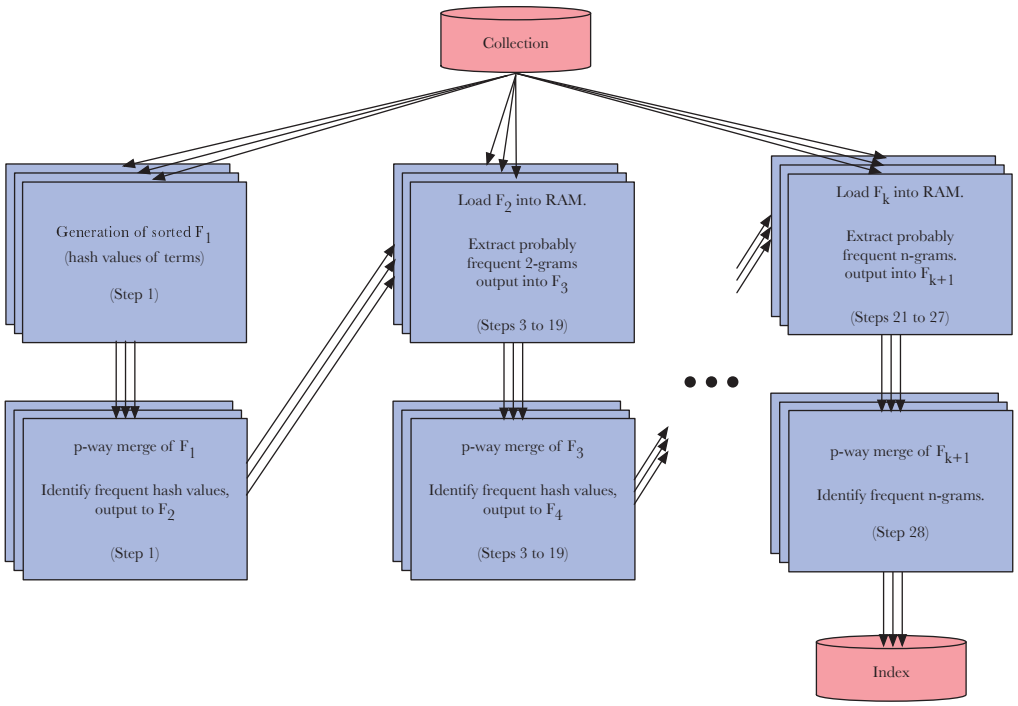


Figure 6.4: Example diagram detailing a distributed implementation of SPEX Multi-Pass and SPEX Log-Pass. Each replicated box represents a set of p processors. Sets of arrows represent the distribution of data across processors.

Algorithm SEQUENCE-BLOCKED WINDOW TWO-PASS

```
1: apply steps 1 to 6 of HASH-BASED WINDOW TWO-PASS to generate file F_2
2: for i ← 1 to M do
3:     for td_j ∈ D_i do
4:         compute surrogate hash value; s ← hash_b(td_j)
5:         append (s, td_j, i) to the in-memory buffer B
6:         if B has reached the memory limit then
7:             sort B
8:             for (s, td_j, i) ∈ B where h ∈ F_2 do
9:                 append (td_j, i, 1) to the output file F_3
10:            end for
11:            B ← { }
12:        end if
13:    end for
14: end for
15: apply steps 6 to 10 of DISK-BASED FREQUENT WINDOW ONE-PASS to the file
    F_3
```

processing node, and, by their randomized nature, cannot be partitioned into segments that correspond to the partitioning of the sequence. So, we assert that each of these algorithms cannot be considered scalable in a distributed sense. In terms of single-processor scalability, each of these methods can be said to be scalable, until the memory limit is reached, but even so, note that the execution time grows as a linear function of both $|C|$ and n. Figures 6.3 and 6.4 show high-level diagrams for distributed implementations of the HASH-BASED WINDOW TWO-PASS and SPEX MULTI-PASS algorithms.

6.2.2.2 Sequential Processing of the Hash Filter

As discussed, HASH-BASED WINDOW TWO-PASS uses a hash filter, S, that (step 14) is presumed to be searchable, and thus available in random-access memory at every processing node. It is clear that this algorithm can not be considered scalable in a parallel sense. We now explore two different methods of avoiding memory requirements, while still retaining the underlying nature of the process.

The first is detailed as Algorithm Sequence-Blocked Window Two-Pass. In this approach, the file F_2 is repeatedly applied to filter subsequences of all extracted term dependencies. In this algorithm, term dependencies are extracted, joined with their hash key, and added to a buffer B whose size is determined by the amount of available main memory. Once the buffer is full, it is sorted by hash key, and then that sorted buffer is merge-intersected with the sorted on-disk file F_2, which contains all of the hash keys of interest. The ones that appear in both B and F_2 are identified, and condensed to form the frequent index structure in the second pass.

In a parallel implementation, each of the p processors is assumed to have local memory, and is able to work with a local buffer, its own full copy of the file F_2, and its own subset of C. Once the frequent term dependencies have been identified, they are sorted back into term dependency order and the index built in the usual manner, at step 15.

The Sequence-Blocked Window Two-Pass approach avoids the need for an unknown amount of memory for S, and replaces that need by a buffer B of predetermined size. However, there is a cost – the smaller the size of B, the more often the file F_2 needs to be merge-intersected against it. So, halving the size of B relative to $|C|$, doubles the overall cost of executing steps 2 to 14.

6.2.2.3 Sorting by Location

The second alternative is to store more information into the intermediate file F_2. The repeated merge-intersect can be eliminated if F_2 is stored in collection location, (ℓ), order rather than hash-value order. To get F_2 into sequence order requires that locations be included in it, in addition to the hash values that are the sort key used at first.

Algorithm Location-Based Window Two-Pass describes the resultant process. Like Algorithm Disk-Based Frequent Window One-Pass, there is very

Algorithm Location-Based Window Two-Pass

1: **for** $i \leftarrow 1$ **to** M **do**
2: **for** $td_j \in D_i$ **do**
3: compute surrogate hash value; $s \leftarrow hash_b(td_j)$
4: $\ell \leftarrow (i, j)$
5: append (s, ℓ) to the output file F_1
6: **end for**
7: **end for**
8: sort F_1, coalescing paired entries (s, ℓ_1) and (s, ℓ_2) to $(s, \ell_1 \cup \ell_2)$ as soon as they are identified
9: **for** $(s, \ell) \in F_1$ **do**
10: **if** $|\ell| \geq h$ **then**
11: append the whole of ℓ to the output file F_2
12: **end if**
13: **end for**
14: sort F_2
15: **for** $(i, j) \in F_2$ **do**
16: extract td_j from document D_i
17: append $(td_j, i, 1)$ to the output file F_3
18: **end for**
19: apply steps 6 to 10 of Hash-Based Window Two-Pass to the file F_3

little memory used at each processing node. File F_1 is a list of hash and location pairs. After the second sorting phase at step 14, file F_2 is a sorted list of "locations of interest" in S. So if there are very few repeated n-grams, file F_2 will be relatively short, and the second "pass" at steps 15 to 16 will skip between locations of interest, only processing (probably) frequent term dependencies in the collection, rather than all of them. That is, Location-Based Window Two-Pass can be expected to have an advantage over Hash-Based Window Two-Pass in terms of execution speed, with the advantage greater when the frequent items are sparser.

To parallelize this method across p processors, the same approach is taken as when parallelizing Disk-Based One-Pass. The source sequence is partitioned into p slightly overlapping subsequences, and each of the processors generates an F_1 file for its subsequence. Those p files are then globally sorted, by creating and exchanging p^2 smaller segments, and carrying out p independent p-way merges. Following that first

sort, each processor then carries out steps 9 to 13 on the sorted segment of F_1 that it holds, before again slicing it into p subsegments and swapping across the p processors, in preparation for the second sort at step 12. Each processor then extracts (steps 15 to 18) each location of interest from the same subsequence of C that it originally worked with in the first pass. Finally, at step 18, the frequent index is built from the file F_3 via a third distributed sorting stage, as described for Disk-Based Frequent Window One-Pass.

6.3 Retrieval Using Frequent Indexes

We have now covered a variety of indexing algorithms that construct frequent indexes of term dependencies. We now consider how these indexes could be used to evaluate the best performing dependency models, as discussed in Chapter 4. This structure can be used in two ways to aid the execution of dependency models. We name these methods "lossy" and "lossless" query evaluation. The key difference between these two methods is how infrequent, and non-indexed, term dependencies are evaluated.

To execute in a lossless mode, a set of frequent index structures of the required term dependencies is supplemented with a positional index of terms. The positional index is used to reconstruct any proximity-based term dependencies that are omitted from the frequent indexes as occurring infrequently. For SDM and WSDM-Int, the algorithms that reconstruct instances of infrequent term dependencies are discussed in Section 5.6.1. This approach ensures that true statistics are used for all term dependency. However, recomputing term dependency features at query time is relatively costly. We also note that it is possible that a fraction of the term dependencies extracted from queries do not occur in the collection. In these cases, positional data must still be read from disk, and evaluated to verify if the term dependency occurs in the collection or not.

In the lossy mode, the query execution model simply ignores missing term dependencies, as it would queries that contain out-of-vocabulary terms. When using the index in this manner, for a given query and dependency model, there is a guarantee that the scores of no more than h documents will be affected, for each infrequent term or term dependency. Unfortunately, if these indexes are used in this lossy manner, then it is reasonable to expect that the h documents that are affected by an infrequent term dependency may be among the most relevant documents for the query.

6.4 Experiments

6.4.1 Ordered Windows

6.4.1.1 Dataset and test environment

We use subsets of the GOV2 and ClueWeb-09-Cat-B collections to test the time and space efficiency of each of these indexing algorithms. The details of each of these collections is provided in Chapter 3. In these experiments, we evaluate the index construction algorithms for the construction of indexes of ordered windows of width 1, containing n terms, or n-grams.

To provide a straightforward basis for the experimental evaluation, each collection was pre-processed to form a sequence of word tokens, stored as 32-bit integers. All header data and document segmentation information were discarded, as was embedded markup and other non-text content such as executable scripts; and then the resultant data was treated as a continuous stream of words. In a practical system, term dependencies would not be permitted to span document boundaries, but the difference is small, and our experiments are realistic. Where smaller test sequences were required, prefixes of this long whole-collection sequence were used. We note that there may be some bias resulting from this method of collection sampling, however, this bias is constant within each experiment, so the comparison of the various techniques remains valid. In all of the experiments reported below $h = 2$ was used, and

the output was thus a list of all term dependencies that appear twice or more in the input sequence, together with their locations as ordinal offsets from the start of the sequence.

The experimental hardware consisted of a cluster of 32 dual-core 64-bit Intel processors with a 3.2 GHz clock speed, and 4 GB of RAM each. The experiments were all configured so that only one core and only 2 GB of memory were used on each processor, with the other core on each machine forced to be idle. Inter-process communication was via a shared network attached file system, with all processors able to write files to the shared disk, and to read the files written by other processors.

Two values of b were used, dependent on whether the hash-filter was required to be memory resident or not. In the Hash-Based Window Two-Pass, SPEX Multi-Pass, and SPEX Log-Pass methods, $b = 32$ was used for the size of the in-memory hash table, and it was stored as a direct-access. With this value the hash-filter contains four billion entries, and at two bits per entry, requires 1 GB of main memory, the largest amount that could be usefully managed on the experimental hardware (and note that SPEX Multi-Pass and SPEX Log-Pass require two such filters to be concurrently available). A filter of this size provides useful discrimination for sequences of up to approximately $|C| = 10^9$ in length for Hash-Based Window Two-Pass, and four to five times larger for SPEX Multi-Pass and SPEX Log-Pass, the difference arising because of the iterative and more selective nature of the insertion policy in the two SPEX Multi-Pass variants. For the hash-filter methods where the filter is stored only on disk – Sequence-Blocked Window Two-Pass and Location-Based Window Two-Pass – a hash function of $b = 56$ bits was used in all experiments, and provided a high degree of discrimination and a low false match rate.

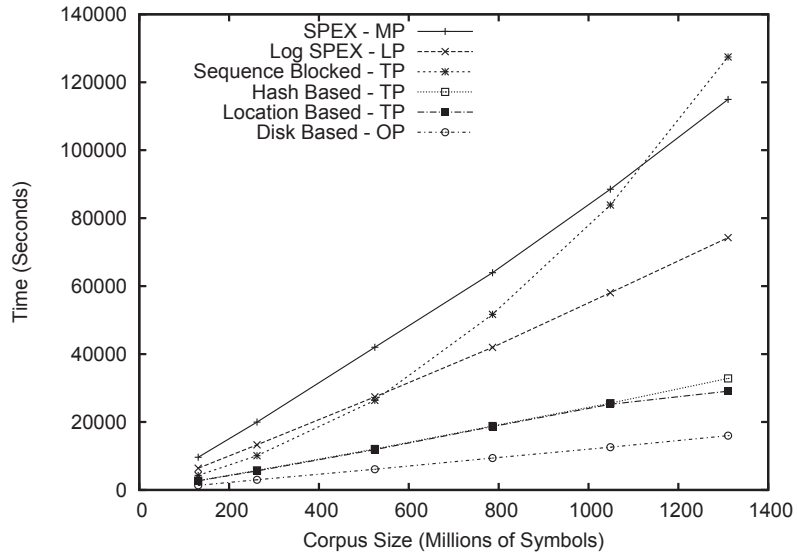


Figure 6.5: Execution time as a function of $|C|$, when computing repeated 8-grams using a single processor. The Sequence-Blocked Window Two-Pass method requires time that grows super-linearly; over this range of $|C|$ the other methods are all essentially linear in the volume of data being processed. All data points in this graph represent an average of 10 timed runs. In this experiment the data sequence used is extracted from the GOV2 TREC collection.

6.4.1.2 Monolithic Processing

Figure 6.5 shows evaluation time as a function of $|C|$, for a single fixed value of $n = 8$, and single-CPU execution. Five of the methods have execution times that grow linearly as a function of $|C|$, with the Disk-Based Frequent Window One-Pass the fastest of the six methods tested. The Sequence-Blocked Window Two-Pass algorithm has performance that grows super-linearly, a consequence of the fixed memory allocation, and the increasing number of blocks B that must be processed against the hash-filter stored in file F_2. The two SPEX Multi-Pass approaches are not competitive, and while they can physically process data files of more than 10^9 symbols, are hindered by the time taken to perform their multiple passes. False matches do not affect the execution time in any significant way, but do result in large temporary disk files F_3 being created. From that point of view, for large $|C|$, it becomes preferable to abandon the SPEX Multi-Pass approach entirely, and

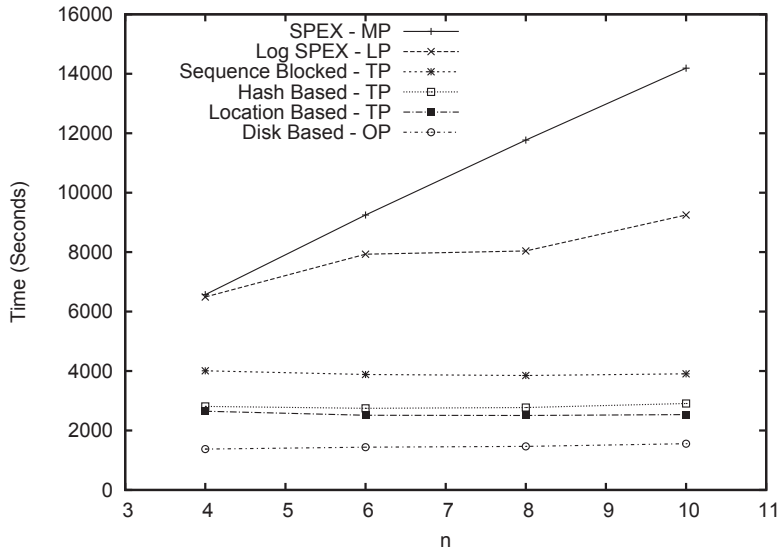


Figure 6.6: Execution time as a function of n, processing a total of $|C| = 125 \times 10^6$ symbols representing English words. Each pass that is made adds considerably to the time taken to identify the repeated n-grams. All data points in this graph represent an average of 10 timed runs. In this experiment the data sequence used is extracted from the GOV2 TREC collection.

simply use the Disk-Based Frequent Window One-Pass method – it uses the same disk space, and is n times faster.

Figure 6.6 then fixes the length of the collection, $|C|$, and varies n, again working on a single CPU. The effect of the multiple passes undertaken by the two SPEX Multi-Pass variants is apparent, and even the second pass through the sequence associated with the hash-filter based variants adds to the running time, meaning that the Disk-Based Frequent Window One-Pass method is the fastest. It, and also the three hash-filter based methods (Hash-Based Window Two-Pass, Sequence-Blocked Window Two-Pass, and Location-Based Window Two-Pass) are relatively insensitive to n, and the cost of evaluating the hash function is not a significant factor in the overall running time, at least over this spectrum of n values.

Measurement of the reading and hashing loop in isolation (steps 1 to 3 of Hash-Based Window Two-Pass, and also used in several of the other approaches) on

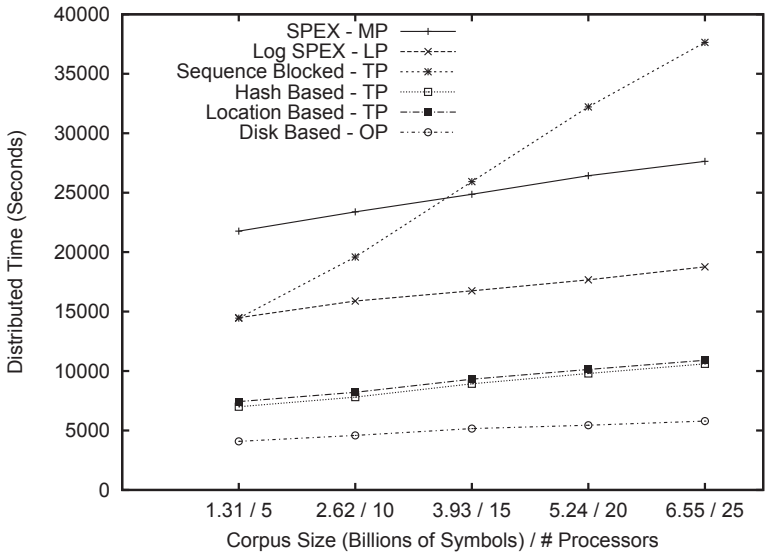


Figure 6.7: Elapsed time required by parallel implementations. All data points in this graph represent an average of 5 timed runs. In this experiment the data sequence used is extracted from the Clueweb-09-Cat-B collection.

a sequence of $|C| = 500 \times 10^6$ symbols showed that there was some small increase in execution time, from around 2,800 seconds when $n = 2$ to around 3,120 seconds when $n = 10$, indicating that the cost of the hash evaluation to generate a $b = 32$-bit value, while varying as n, has only a small effect on overall execution time. Without the call to the hashing function, the same loop required 2,100 seconds at $n = 2$ and 2,400 seconds when $n = 10$, confirming that the `md5` hash routine is a non-trivial, but also non-dominant part of the first processing loop, and is relatively unaffected by the length of the n-gram it is acting on.

6.4.1.3 Parallel Processing

A critical claim in this chapter is that the Location-Based Window Two-Pass approach is scalable, and can readily be implemented across a cluster of computers in order to deal with very large collections. Figure 6.7 demonstrates the validity of that claim. To generate this graph different length prefixes of the ClueWeb-09-Cat-B collection were taken, and split over 5, 10, 15, 20, and then 25 processing nodes, with

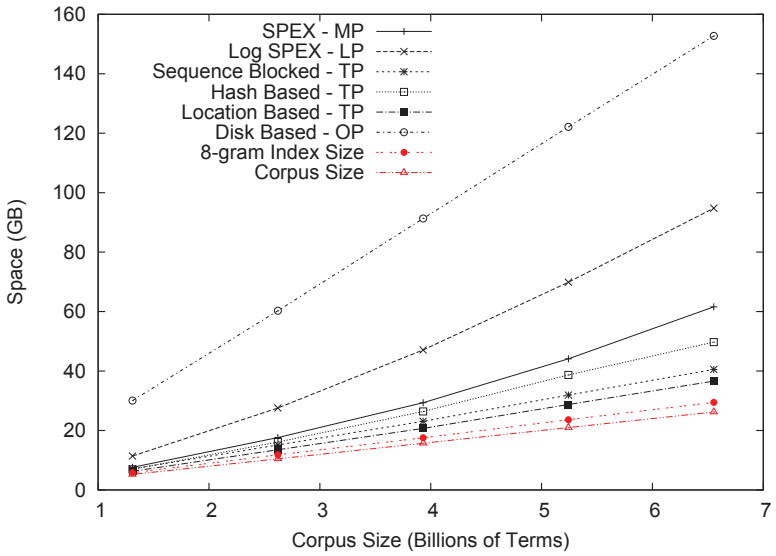


Figure 6.8: Peak temporary disk space required as a function of $|C|$, when constructing indexes of frequent 8-grams. In this experiment the data sequence used is extracted from the Clueweb-09-Cat-B collection.

the prefix lengths varied in proportion to the number of processing nodes involved. In Figure 6.7, each processing node hosts a collection subsequence of approximately 270×10^6 terms. With this experimental design, elapsed time for a scalable algorithm should be either constant, or slightly growing if there is a logarithmic overhead on a per-processor basis.

Five of the methods show the required trend, while the sixth, for the Sequence-Blocked Window Two-Pass method, does not – as was anticipated above, it cannot be regarded as being scalable. The Hash-Based Window Two-Pass and the two SPEX Multi-Pass variants do scale in a "CPU cost only" sense as the sequence length increases, and the breakdown in the performance of the hash-filter as $|C|$ increases is not dramatic in terms of execution time. However, as is discussed in subsequent experiments, they are not as well behaved in terms of disk space requirements.

6.4.1.4 Intermediate disk space requirements

Figure 6.8 plots the peak temporary disk space required by each method, as a function of $|C|$ when $n = 8$, for collections of sizes 1×10^9 terms to 6×10^9 terms. The Disk-Based Frequent Window One-Pass method is very expensive, a consequence of the fact that around $(n + 1)N$ words are required in the temporary file F. It takes more than twice the peak disk space of the hash-filter methods.

The next most expensive method is the Hash-Based Window Two-Pass approach. The issue raised by the $b = 32$ bit hash size is now clearly apparent; hashing a billion or more objects into a $b = 32$ table yields a significant number of false matches, and all of these create entries in the F_3 file. This problem is exacerbated as more and more symbols are indexed. For smaller values of N (not shown in the graph), Hash-Based Window Two-Pass is more competitive, but like the SPEX Multi-Pass variants, when $|C|$ is large it is simpler and more efficient to revert to the Disk-Based Frequent Window One-Pass approach.

The next two data points correspond to the SPEX Log-Pass and SPEX Multi-Pass algorithms. These methods do not write a hash-filter to disk at all, but both generate a large F_3 file even for small $|C|$, primarily because the hash-filter that is constructed by the incremental process is not completely precise – it generates many false matches. This stems from the use of the $b = 32$ bit hash size throughout these algorithms. It is clear that for a fixed, maximal hash size, as the collection size increases, the effectiveness of the final filter is slowly diminished.

The Location-Based Window Two-Pass method performs well, despite requiring more space for the F_1 file than does the Hash-Based Window Two-Pass approach (recall that the difference is that a location offset is stored with each $b = 56$ bit hash value, rather than a 2-bit counter). Storing the first temporary file F_1 is the dominant space cost for this amount of data and this degree of reuse. That is, compared to Disk-Based Frequent Window One-Pass, Algorithm Location-

Based Window Two-Pass requires significantly less disk space, and executes less than two times more slowly.

The two curves at the bottom of Figure 6.8 show, respectively, the cost of storing the eventual 8-gram frequent index, where $h = 2$, computed assuming that each 8-gram requires eight words to store the gram, plus one address per gram occurrence; and the actual sequence size, stored one value per word. The peak temporary disk space requirement for all of the methods still exceeds the final index size, and also exceeds the corpus size. It may be that this gap represents room for further improvement, but it seems likely that a fundamentally different paradigm will be required before that possibility can be realized. Note also that at least some of the apparent gap is a consequence of the data being distributed across nodes – the "coalescing of like entries during sorting" steps of the various algorithms becomes less effective as the data is partitioned across more machines.

6.4.2 Unordered Windows

This analysis demonstrates that two algorithms can be considered both efficient and scalable in space and time requirements, by our definitions of monolithic and parallel scalability, when indexing n-grams (ordered windows of width 1, containing n terms). Following these observations, we now investigate the efficiency of these two indexing algorithms for the construction of frequent indexes of unordered windows of width 8, containing 2 terms. These experiments are intended to verify if these algorithms continue to be considered scalable when extracting unordered window dependencies. Future work will include investigating the application of these techniques to larger windows, and unordered windows containing more than 2 terms.

6.4.2.1 Monolithic Processing

First, we seek to determine if the indexing algorithms presented above for n-gram data remain scalable, in both time and space, when used to index unordered window

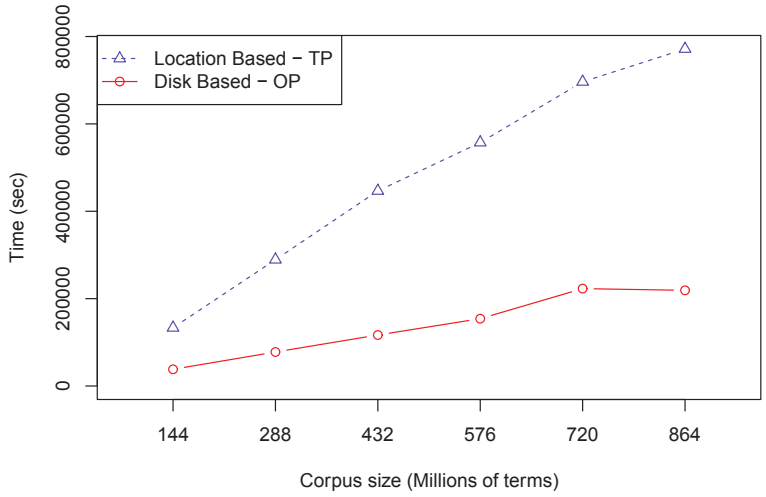


Figure 6.9: Elapsed time required by parallel implementations. All data points in this graph represent an average of 10 timed runs. In this experiment the data sequence used is extracted from the TREC GOV2 collection.

data in a monolithic setting. In this experiment, we use the TREC GOV2 collection. Similar to previous experiments, terms in this dataset have been enumerated to 32 bit integers. During indexing, we extract unordered windows of width 8, containing 2 terms from each document. The collection is divided into a number of subsequences of around 144 Million terms. This corresponds to around 1 trillion unordered windows of this type. We the measure the time required to construct indexes of sets of these subsequences.

Figure 6.9 shows the average time required to construct a frequent index of unordered windows, as the size of the corpus, $|C|$, is increased, using both Disk-Based Frequent Window One-Pass and Location-Based Window Two-Pass algorithms. Each data point is the average of 10 repetitions. We observe that both algorithms are scalable when indexing unordered windows of width 8, containing 2 terms. Again, the Disk-Based Frequent Window One-Pass algorithm is observed to be faster than the Location-Based Window Two-Pass algorithm.

Intermediate space requirements were also measured for this experiment. We have previously observed that the space requirements of the filter structure (F_1) used by Location-Based Window Two-Pass algorithm are smaller than the postings extracted (F_3) during the second pass over the collection. For this type of unordered windows, the space requirements of the filter file, F_1 are smaller than F_3, but only minimal space saving are observed. Where the final index file for a collection containing 144 million terms occupies around 14 GB, the intermediate space requirements for the Disk-Based Frequent Window One-Pass algorithm is just over 15 GB. The Location-Based Window Two-Pass algorithm requires almost 15 GB to store F_1. The relatively large fraction of retained items in the vocabulary of the frequent index is the cause of these relatively modest space savings. As the threshold h, is increased, the number of frequent windows drops, thereby dropping the size of the final index, and increasing the relative space savings that can be achieved through the Location-Based Window Two-Pass method. The final space requirements of frequent indexes of unordered windows are further investigated below, in Section 6.5.

6.4.2.2 Parallel Processing

Next, we investigate if the best performing algorithms for n-gram data remain scalable, in a distributed sense, when used to index unordered window data. Recall that in order to be considered scalable in a distributed sense, the time required to construct an index for a specific collection must remain constant, or grow only slowly, as the collection, and the number of processors are increased at the same rate. Identical to the previous experiment, we use the TREC GOV2 collection. Similar to previous experiments, terms in this dataset have been enumerated to 32 bit integers. Again, we extract unordered windows of width 8, containing 2 terms from each document. The collection is divided into a number of subsequences of around 144 Million terms.

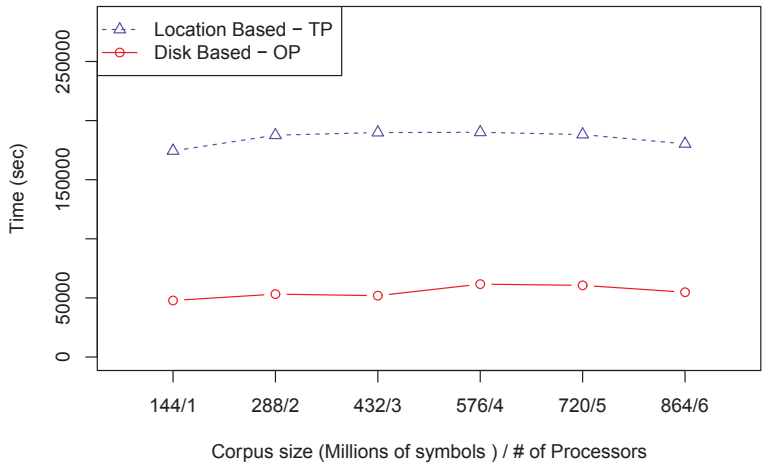
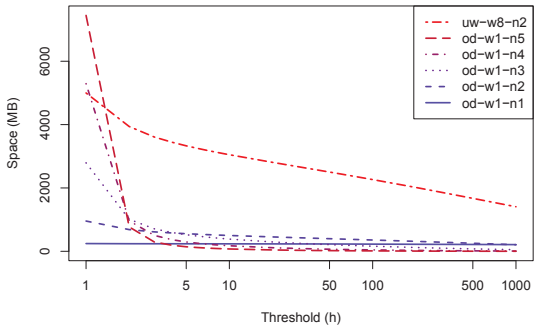


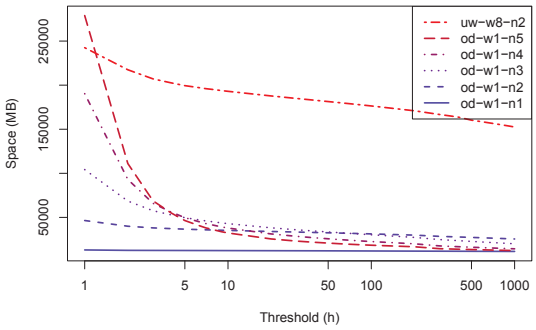
Figure 6.10: Elapsed time required by parallel implementations. All data points in this graph represent an average of 10 timed runs. In this experiment the data sequence used is extracted from the TREC GOV2 collection.

This corresponds to around 1 Trillion unordered windows of this type. We measure the time required to index a set of p subsequences using p processors.

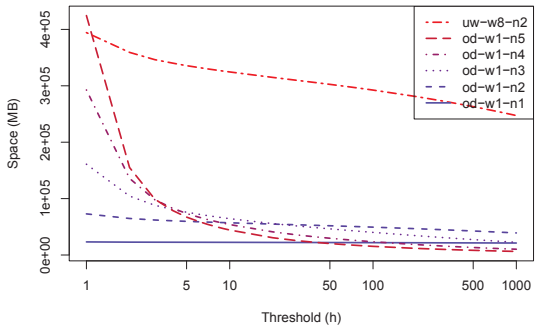
Figure 6.9 shows the time required to construct a frequent index of unordered windows, as the size of the subsequence is increased, and the number of processors is increased, using both Disk-Based Frequent Window One-Pass and Location-Based Window Two-Pass algorithms. Each data point is the average of 10 repetitions. As the size of the collection and the number of processors are increased, the time required to construct an index using either of these algorithms remains approximately constant. In accordance with previous results, this data indicates that both the Disk-Based Frequent Window One-Pass and the Location-Based Window Two-Pass algorithms can be considered scalable in a parallel processing time.



(a) Robust-04



(b) GOV2



(c) Clueweb-09-Cat-B

Figure 6.11: Space requirements for frequent indexes of three collections, for 6 different types of windows, over a range of threshold values.

Table 6.2: Space usage comparison between full indexes, and frequent indexes, with threshold parameter $h = 5$.

Window Type	Full Index		Frequent Index ($h = 5$)	
	Vocab. (MB)	Post. (MB)	Vocab. (MB)	Post. (MB)
Robust-04				
od-w1-n1	4.42	239	1.22 (−72.4%)	234 (−2.09%)
od-w1-n2	248	707	36.6 (−85.2%)	519 (−26.6%)
od-w1-n3	1,490	1,302	89.0 (−94.0%)	434 (−66.7%)
od-w1-n4	3,485	1,806	80.2 (−97.7%)	215 (−88.1%)
od-w1-n5	5,378	2,068	51.1 (−99.0%)	92.3 (−95.5%)
uw-w8-n2	1,103	3,895	206 (−81.3%)	3,123 (−19.8%)
GOV2				
od-w1-n1	426	12,660	58.4 (−86.3%)	12,310 (−2.76%)
od-w1-n2	6,062	40,393	939 (−84.5%)	35,710 (−11.6%)
od-w1-n3	35,915	68,319	4,515 (−87.4%)	45,186 (−33.8%)
od-w1-n4	97,748	92,614	8,194 (−91.6%)	41,490 (−55.2%)
od-w1-n5	170,069	108,868	10,337 (−93.9%)	36,093 (−66.8%)
uw-w8-n2	27,779	214,717	4,560 (−83.5%)	194,882 (−9.2%)
Clueweb-09-Cat-B				
od-w1-n1	418	22,787	90.9 (−78.2%)	22,424 (−1.59%)
od-w1-n2	8,665	64,238	1,809 (−79.1%)	57,845 (−9.95%)
od-w1-n3	58,608	102,353	8,235 (−85.9%)	67,628 (−33.9%)
od-w1-n4	157,664	135,006	14,603 (−90.7%)	58,969 (−56.3%)
od-w1-n5	268,494	156,047	18,230 (−93.2%)	48,947 (−68.6%)
uw-w8-n2	39,955	354,360	8,502 (−78.7%)	327,176 (−7.67%)

6.5 Space requirements

In this section, we investigate the impact the threshold parameter h, has on the final index space requirements of frequent indexes of ordered windows of width 1, containing 1 to 5 terms (1- to 5-grams), and frequent indexes of unordered windows of width 8, containing 2 terms. For this experiment, we construct frequent indexes of three collections; Robust-04, GOV2, and Clueweb-09-Cat-B. Unlike previous experiments, we do not use enumerated terms, instead terms are stored as UTF-8 strings. Similar to previous experiments, *vbyte* encoding is used to compress the posting list data. Note that this data can also be estimated from the distributions of ordered and unordered window presented in Chapter 5.

Figure 6.11 shows space requirements for a frequent indexes of ordered and unordered windows, for each collection. Note that the threshold values are plotted in log-scale. This data shows that the largest reductions in space requirements occur for the smallest threshold values. Interestingly, two-term, unordered windows of width 8 do not show the same degree of space savings as ordered windows of width 1.

Table 6.2 displays a comparison of the space requirements of full index with the space requirements of frequent indexes with threshold parameter $h = 5$. The majority of space savings achieved by the frequent index is vocabulary data. In comparison to a full index of term dependencies, we observe that between 70% and 99% of the vocabulary space requirements are reduced by this type of index. Some significant space savings are also observed for the postings data of larger many-term ordered windows. Depending on the type of window indexed, the postings data is reduced by between 1.5% and 67%.

This data helps account for the high space requirements of the unordered windows observed in Figure 6.11. We can see that the vocabulary data requirements are reduced to a similar fraction as for two-term ordered windows, however, the postings data requires considerably more space. This is expected, the total number of instances

of two-term unordered window, of width 8, is around 7 times larger than the number of terms in the collection.

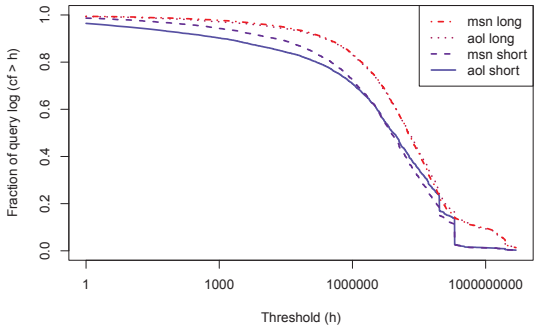
This data helps detail the relationship between vocabulary and postings data, for the tail of the vocabulary. That is, the tail of the vocabulary accounts for a large fraction of the vocabulary space requirements, but only a small fraction of the posting list space requirements. It is reasonable to expect that discarding the most frequent term dependencies would more dramatically affect the space requirements.

6.6 Retrieval Experiments

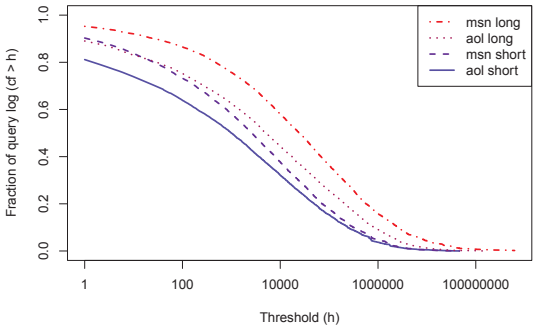
6.6.1 Query Log Analysis

We now investigate available query log data to determine the fraction of features extracted from queries that may be discarded by various threshold settings. We perform this analysis over the two available query logs. We measure collection frequency using the Clueweb-09-Cat-A collection. We use the MSN and AOL query log data described in Chapter 3 in this experiment. Each query log is separated into two categories; short and long queries. Short queries contain at most 3 words and long queries contain between 4 and 12 words. This separation is based on a query log analysis (Bendersky and Croft, 2009). Importantly, Bendersky and Croft (2009) show that longer queries can have very different properties than shorter queries.

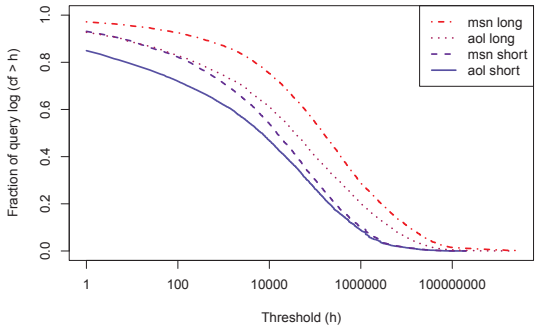
In this experiment, we assume that the Uni+O23+U23 variant of the sequential dependency models, investigated in Chapter 4, is used to execute each query. We extract all sequential sets of 1-to-3 terms from each query. $10,000$ term set instances are then uniformly-at-random sampled, for each set size, $(1 \leq n \leq 3)$. Each set of terms is then matched to instances in the Clueweb-09-Cat-A collection, as both an ordered and an unordered window. The width of each ordered window is set to one, and the width of each unordered window is set to 4 times the number of terms in the set.



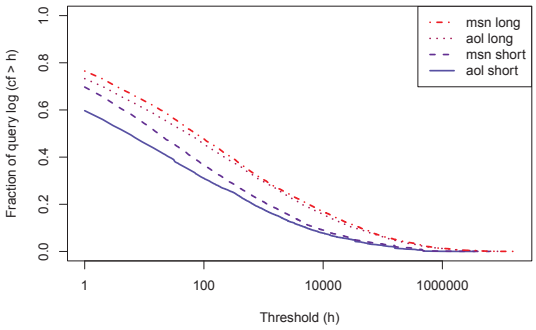
(a) Terms



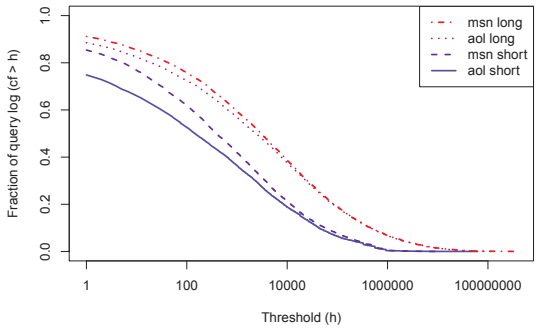
(b) `od-w1-n2`



(c) `uw-w8-n2`



(d) `od-w1-n3`



(e) `uw-w12-n3`

Figure 6.12: Query log analysis showing the fraction of terms and windows that would be discarded through a frequent index policy, for the MSN and AOL query logs, for a range of threshold values.

Figure 6.12 shows the fraction of the each query log sample that passes a range of frequency threshold values. This data measures the degree to which each of these retrieval model features would be impacted by the use of frequent indexes, for a range of threshold values. Note that the x-axis of each graph is plotted in log scale.

Except for ordered windows of three-term sets, each of these graphs shows a plateau for a range of low threshold values. This indicates that a threshold value within this range would affect very few of the corresponding type of query feature. So, it is reasonable to expect that very few queries would be directly affected by the use of frequent indexes. The majority of query features occur either more frequently in the collection, or do not occur at all in the collection.

Further, this data shows that a large fraction of the larger window features ($n = 3$), do not occur in the collection. This observation has important consequences for the lossless execution model discussed in Section 6.3. It shows that the percentage of window features that do not occur in the collection, but must still be recomputed from positional data, could approach 40%.

Next, we inspect a small sample of the least frequent, but non-zero frequency, sets of terms extracted from each query log. Table 6.3 and 6.4 shows a sample of ten infrequent sets of terms, from each query log sample. All displayed examples occur fewer than 20 times in Clueweb-09-Cat-A, placing these instances in the plateau observed in the aggregate data.

First, in the creation of this example data, we observe a disproportionate presence of partial urls in the infrequent sets of terms, and sets of terms. Both MSN and AOL query logs contain a very large number of queries that appear to be urls or partial urls. In our index of Clueweb-09, url data (`<a href=‘‘url’’>`) is not indexed, this may partially account for the low observed frequency values for url components. In this example data, we do not provide examples of this type of term or set of terms.

Table 6.3: Example infrequent terms and pairs of terms from query-log samples.

MSN		AOL	
short	long	short	long
Terms			
confliguration	stenceils	dayspringe	trevbain
noveltoy	elevaction	adawre	cirruclation
ajslick	dolejuice	bearload	palacd
bluthooth	volisia	clutchflite	sergoune
kerosean	triathlalon	turatus	lienson
ebaus	genoraters	aumsworld	hazelpeaches
americaproducts	opinionport	kapshandy	aleania
lyndard	pregancey	afracans	hondrus
zhingles	southcast	caverens	financialbank
valpanaro	csncer	kitcheaid	starkcounty
od-w1-n2			
omen apple	tribbett lyrics	shinkendo ny	saturn used
medciare gov	emilee jo	galveston rent	red bassett
madera chih	ucsd compliance	microsoft publiser	gallery press
avon yarns	roudner coral	frank alteri	boy amputation
perry hoskins	dermatoligists in	tahitian orchid	guide pyramide
mercedes seattle	ranch delano	marvello country	lyrics rockabilly
motels lacrosse	rota tiller	yellow mccoy	county sherriff
paula falcone	ekg fast	howe obituary	sherock holmes
ramada hayward	wings greensboro	thornton sargent	epcot beach
amercan west	alakanuk flooding	arkansas travlers	roanoke virginia
uw-w8-n2			
godfather soundboard	mandalay reality	amforal tablet	aspen stevens
wachovia payformance	nissian skyline	weightloss supplaments	timothy vakoc
lasar institute	mycampus phoenix	www annamariaisland	city marker
seqoia camping	square ennix	van tichelin	life paintig
volkwagen eos	atrlantic city	rajitas peludas	aloha grafics
japanese paperdolls	thermal camra	kdp nu	tom chapin
leaf tannen	atlacatl monument	image quix	saturn used
calado wine	brant greensboro	dentisti com	timothy vakoc
firstbank crowley	marsha hockman	frank morino	city marker
enamel fixall	appreation gifts	asino resort	life paintig

Table 6.4: Example infrequent sets of three-terms from query-log samples.

MSN		AOL	
short	long	short	long
od-w1-n3			
bakersfield football camps	beans using vinegar	mechanicsville white pages	airborne deaths in
hampton inn md	juice mountain man	spokane general store	of jamaican dancer
aaa aproved hotels	sore like irritation	bmw engine rebuilt	the sea sparknotes
anderson copper 360	water heater complaints	woody allen theme	ms day use
lake tahoe termite	black magic ozonator	water grain mill	fabric store coupons
ottery barn kids	star named angel	1979 swimming pools	style warded lock
diamond mines wa	a origami owl	grannys speed shop	left outer hip
standard baffle design	bath room rugs	food network recipies	preschool birthday cake
gentlemen fight clubs	air conditioning carmel	the diamond dr	pico rivera police
x ray technition	book binding near	cruise payment plan	tractor model 770
uw-w12-n3			
hawaiian silver jewelery	1337 coronel street	philadelphia docket report	power steering willys
eureka sanitaire sc3684	points sheridan wichita	napolean dynamite soundboard	diamond dome glazed
jobsource of indiana	sempervirens tiny tower	strenght in chinese	steffans office equipment
rost motor company	jacob frederick bunn	jacob frederick bunn	brice prairie canoe
mangosteen and autism	builda bear workshop	raising baby fawns	circut court access
primera mano english	conagra pioneer farm	builda bear workshop	angelas harbor college
caprine disease texas	intertherm hot water	modern shinkendo ny	ferm moisture retainer
willaim van architect	of shamar moore	pastor steve mcdonald	shore huntington collection
kinzey lake catherine	fraser martin buchanan	pillow case burglar	ed waring dance
pastor steve mcdonald	mcormick and schmicks	charlseton south carolina	factors to consdier

In this displayed example of infrequent terms, we observe that a number of the infrequent terms, and sets of terms, are incorrectly spelt, or appear to be two words concatenated together. In both of these cases, these terms are unlikely to be effective query features. While it is not clear which query suggestion algorithms were used when these query logs were collected, it is possible that improved query suggestion or spelling correction algorithms would reduce the instances of these terms in the query log.

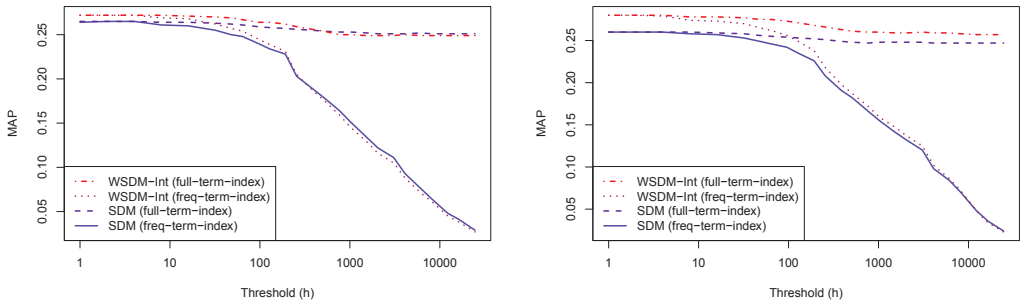
Infrequent sets of two and three terms contain fewer spelling or concatenation errors. We observe that many of the infrequent 2- and 3-term ordered windows, become frequent when matched as unordered windows. Further, we note that several infrequent 3-term ordered windows seem intuitively to be good query features. It is possible that documents that focus on these topics may not exist in this document collection.

It is important to acknowledge that the data available in these two query logs is far from perfect for this experiment; each set of queries has been filtered and personal data has been obfuscated using a variety of unknown operations. These processes may directly influence the distribution of frequent and infrequent windows present in the query-log data. However, the techniques used in this section would be applicable to a more complete query log.

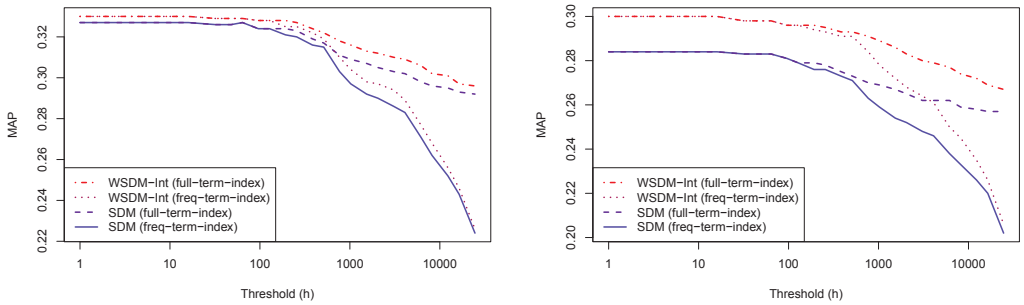
6.6.2 Retrieval Effectiveness using Frequent-Indexes

In Section 6.3, above, we detail two methods of using frequent indexes in a retrieval system. We now test the effect of this type of index, used in a lossy manner, on two of the strongest performing dependency models identified in Chapter 4, SDM and WSDM-Int.

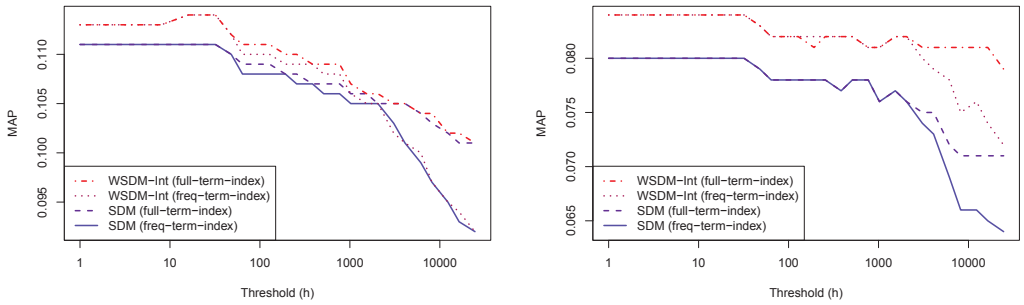
We simulate the effect of frequent indexes, by applying a threshold value to discard a subset of the features used in these retrieval models (terms, ordered windows and



(a) Robust-04, title and description topics

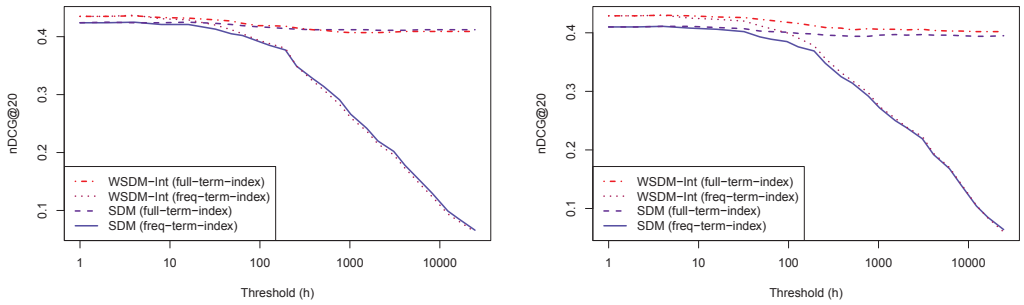


(b) GOV2, title and description topics

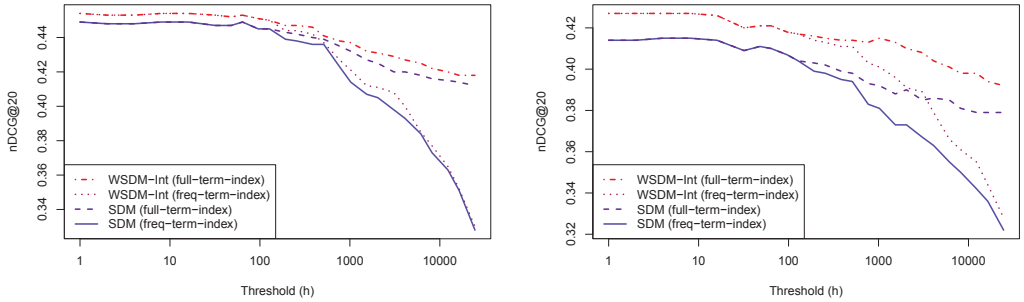


(c) Clueweb-09-Cat-B, title and description topics

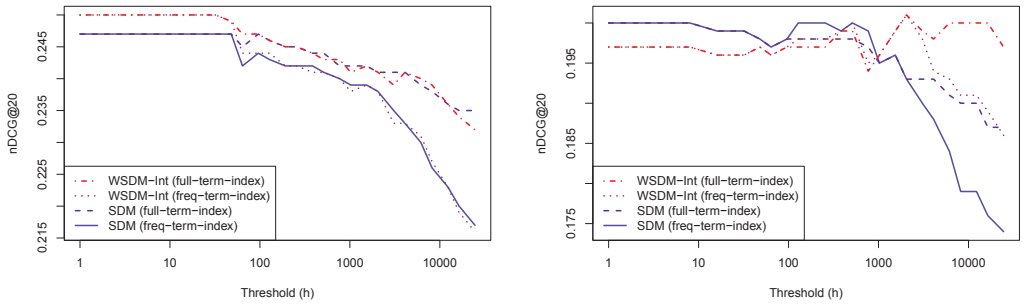
Figure 6.13: Changes in measured MAP, when using frequent indexes, for a range of threshold values. Results for title (left) and description (right) topics are both shown. "Full-term-index" indicates that a frequent index was not used for term features, just for dependency features. "Freq-term-index" indicates that term features were also stored in a frequent index.



(a) Robust-04, title and description topics



(b) GOV2, title and description topics



(c) Clueweb-09-Cat-B, title and description topics

Figure 6.14: Changes in measured nDCG@20, when using frequent indexes, for a range of threshold values. Results for title (left) and description (right) topics are both shown. "Full-term-index" indicates that a frequent index was not used for term features, just for dependency features. "Freq-term-index" indicates that term features were also stored in a frequent index.

unordered windows). A query feature is omitted, if the collection frequency of the feature is lower than the threshold, just as it would be omitted in a frequent index structure. We consider two possible scenarios in this experiment. First, we assume that the search engine has access to a full index of terms, and a set of frequent indexes of window features. In results, this scenario is labeled "full-term-index". Second, we assume that all retrieval features, including terms, are stored in frequent indexes. In results, this scenario is labeled "freq-term-index".

We use three TREC collections, Robust-04, GOV2 and Clueweb-09-Cat-B, as used in Chapter 4. Model parameters are set as the average parameter settings across the 5 folds, for each model, for each collection, and both topic titles and descriptions. Figures 6.13 and 6.14 show retrieval performance, using frequent indexes, with a wide range of threshold values.

Generally, retrieval performance degrades slowly as the threshold parameter is increased. This data shows that relatively small threshold values can result in no change in retrieval performance. Larger threshold values are observed to significantly degrade retrieval performance. Further, differences caused by the use of full indexes for terms are only observed for larger thresholds.

We test for significant differences from the original performance of each model using Fisher's randomization test ($\alpha = 0.05$). For the Robust-04 collection, we observe that threshold values $h > 32$ displayed a significant change from the baseline method, across both title and description topics, and for both MAP and nDCG@20. Similarly, threshold values above 128, for the GOV2 collection, and 256, for the Clueweb-09-Cat-B collection, significantly degrade retrieval performance.

It is important to note that this experiment uses a relatively small number of TREC curated queries, a total of 450. It is possible, or even likely, that a larger set of queries, with document clicks or relevance data, would produce different observations about significant differences. However, the overall trends are not expected to change.

6.7 Summary

In this chapter, we have explored the problem of constructing an index of frequent term dependencies, for large English corpora. We then investigated the impact this structure has on retrieval performance.

Our new Location-Based Window Two-Pass algorithm provides a useful blend of attributes, in that it requires less than half the amount of temporary disk space of the Disk-Based Frequent Window One-Pass approach, while requiring less than twice as much processing time. This represents a significant benefit in terms of practical utility. We demonstrated that this approach can readily be adapted for use across a cluster of computers, and is scalable in this distributed sense, a virtue that more than compensates for its slightly slower execution speed. We also verified that both of these algorithms can be considered scalable in both monolithic and parallel senses, for the problem of indexing frequent unordered windows.

We investigated the relationship between the threshold parameter and the space required by the index structure. We observe that the largest space reductions, relative to a full index, occurs at the lowest threshold values $h < 5$. We also analyzed how the frequent index structure affects retrieval effectiveness through a query log analysis, and by studying the change in effectiveness for annotated TREC topics. The query log analysis showed that only a small fraction of queries would be affected by the frequent index structure, for a range of different term dependencies. Finally, the retrieval effectiveness experiments show that threshold values $h < 100$, generally ensure that retrieval effectiveness is unchanged relative to a full index of term dependencies.

In Section 6.2.2.1, we argue that the Hash-Based Window Two-Pass algorithm is not scalable in a parallel sense. This is because it requires that a hash table large enough to store a frequency for each window in the collection must be stored in memory on each computing node. To ensure minimal collisions, the hash table must grow with the size of the collection, so, this approach is not scalable in a parallel

sense. An alternative, that we didn't consider in this chapter, is to use a **CountMin** sketch in place of the hash table. We experiment with this structure in Chapter 7. We observe that the memory requirements of this structure grows sub-linearly with the size of the collection. Importantly, this observation could make the Hash-Based Window Two-Pass algorithm feasible for very large collections, even with limited per-processor memory space.

In our evaluation of the impact of the frequent index on retrieval effectiveness, we use two available query logs, and all TREC topics used in Chapter 4. We know that the query logs used have been filtered by an unknown process. Similarly, the TREC topics have been manually curated. Both processes may reduce the number of infrequent terms and windows present. This may artificially reduce the effect of discarding infrequent items from the collection. A large scale click log, that has not been filtered would provide a more accurate evaluation of the effect the frequent index has on retrieval performance.

An important condition of this index is that it must be constructed over a static, or only slowly updating collection. Each time a document is added to the frequent index, an infrequent term dependency may become frequent. However, since the index does not retain data about these items, it is not possible to determine when a specific term dependencies becomes frequent, as shown in Figure 6.1. We have performed a previous study on this topic. The outcome of this research is that an external structure is required to retain (probabilistic) statistics for all terms and term dependencies in the collection. We determined that an appropriate structure is a sketch. Essentially, the filter structure stored in F_1, in the Hash-Based Window Two-Pass algorithm, is replaced by an in-memory **CountMin** sketch. There are some issues with this approach: the sketch must be stored in memory and cannot be extended as the collection grows. There are several options available to address these issues. First, documents can be assigned a time-to-live, and are removed from the index as the time-

to-live expires. Second, frequent-shard indexes could be constructed, as a particular sketch approaches saturation, the associated frequent index is flushed to disk, and a new index constructed. This approach approximately corresponds to the Lossy-Counting algorithm (Manku and Motwani, 2002) used in stream processing. In future work could include an investigation the effect of constructing frequent-index-shards, for large collections, where a term instance must be frequent in a local shard, to be included in the index. Further, the investigation should include an evaluation of the trade-offs involved in adding and removing documents dynamically to construct a set of index structures that spans a specific time window.

CHAPTER 7

SKETCH INDEX

7.1 Introduction

In this chapter, we present an indexing structure that uses data stream sketching techniques to estimate term dependency statistics. Again, we focus on the ordered and unordered window features that are required by SDM and WSDM-Internal, the strongest performing dependency models identified in Chapter 4.

The experiments conducted in this chapter will focus on n-grams, which are identical to the ordered window features used by these retrieval models. We also claim that this structure is applicable to unordered windows, and other types of term dependencies. This is based on observations made in Chapter 5, showing that the vocabulary growth rates, and vocabulary skew for unordered windows of various widths are not fundamentally different to those observed for n-grams. We leave the empirical investigation of sketch indexes of other types of term dependencies to future work.

The sketch index is derived from a **CountMin** sketch (Cormode and Muthukrishnan, 2005a), and designed to minimize space usage while still producing accurate statistical estimates. This strategy also ensures that the space required by the index is largely independent of the size of the vocabulary of indexed term dependencies, while still supporting efficient query processing.

Conceptually, our summary sketch is an (ϵ, δ)-approximation of a full inverted index structure. So, the index representation is capable of estimating collection and document statistics for indexed term dependencies with bounded error. We show that the relative error of extracted term dependency statistics can be probabilistically

bounded and describe how these bounds minimize the space requirements of the structure in practice. We establish that the retrieval efficiency of the sketch index is comparable to full indexes, and notably faster than positional or next-word indexes. Finally, our experiments demonstrate that our estimator does not significantly alter the query effectiveness when using state-of-the-art n-gram models. This work has been previously presented in a poster paper (Huston et al., 2012), and an extended study of the structure has also been published (Huston et al., 2014).

The major contributions in this chapter include:

- presentation of the sketch index data structure;
- the empirical evaluation of the accuracy of the statistics extracted from the sketch index;
- analysis of relationship between retrieval effectiveness and space requirements of the sketch structure; and
- an empirical evaluation of retrieval efficiency benefits over baseline index structures, afforded by the sketch index.

7.2 Sketching Data Streams

Algorithms for approximating the frequency of items in a collection or a stream have advanced dramatically in the last twenty years (Cormode and Hadjieleftheriou, 2008, 2010). This line of research is based on the streaming model of computation, and has widespread applications in networking, databases, and data mining (Muthukrishnan, 2005). Much of the work in the networking community using these tools has focused on identifying "heavy-hitters", or top-k items (see (Berinde et al., 2009) or (Cormode and Hadjieleftheriou, 2008) and the references therein). If only the k most frequent items must be accurately estimated, counter-based approaches work well in practice. However, counter-based methods are generally not sufficient if estimates for

all the items in a stream are desirable since the number of counters is limited to the top-k subset of items in the stream. For frequency estimation of any item in a stream, various "sketching" methods are an appropriate alternative.

Here, we limit our discussion to sketching algorithms as these data structures are able to bound the allowable error of approximation for all items in a stream. A *sketch* is a hash-based data structure that represents a linear projection of the streaming input. Two general approaches to sketching are present in the literature: **AMS** and **CountMin**.

The **AMS** sketch was first proposed by Alon et al. (1999) to estimate the second frequency moment ($\mathcal{F}_2$), relative the collection size ($|\mathcal{C}|$), with error $\epsilon\sqrt{\mathcal{F}_2} \leq \epsilon|\mathcal{C}|$ with probability at least $1 - \delta$ for a sketch using $\mathcal{O}(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ bits of space. However, the original representation of **AMS** is not efficient in practice since the whole sketch must be updated for each new item.

To address this shortcoming, Charikar et al. (2002) proposed a modification that ensures each update only affects a small subset of the entire summary. Charikar et al. (2002) refer to this approach as a **CountSketch**. The key idea of a **CountSketch** is to create an array of $r \times w$ counters, with independent hash functions for each row r. The hash functions map each update to set of counters, one in each row r. In addition, another independent hash function maps the value $\{-1, +1\}$ to each update. This approach ensures that collisions over the entire distribution are likely to cancel out. Increasing the number of rows used (r) lowers δ. So, to match the same ϵ and δ bounds of the **AMS**, the values $r = \log \frac{4}{\delta}$ and $w = \mathcal{O}(\frac{1}{\epsilon^2})$ are used. Using these parameters, the space bound for the **CountSketch** is now identical to **AMS**, but update time is reduced to $\mathcal{O}(\log \frac{1}{\delta})$.

Another sketching alternative was recently proposed by Cormode and Muthukrishnan (2005a). The **CountMin** sketch is similar in spirit to **CountSketch**, in that the method uses an array of $r \times w$ counters. The key difference is the omission of

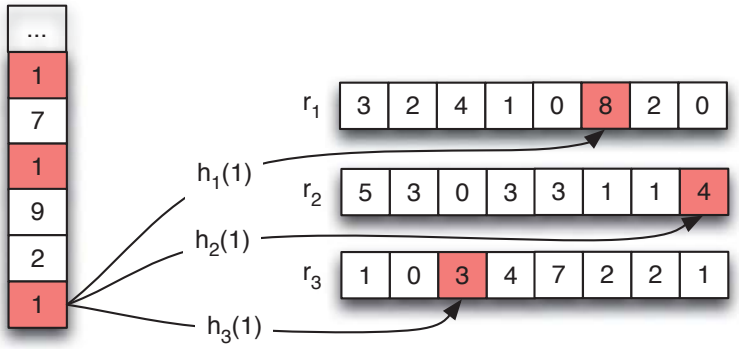


Figure 7.1: Example **CountMin** Sketch containing frequency data for a stream of integers. Where the highlighted integer, 1, is hashed to each of the highlighted cells. The frequency of 1 in this stream is estimated as the minimum value of the highlighted cells $f_1 = 3$.

the secondary hashing function which maps $\{-1, +1\}$ onto each update. Instead, **CountMin** always increments each counter. In streams which do not include deletions, this ensures that the frequency of any item $f(i)$ in the sketch is an overestimate. The expected number of collisions for i on any row is $\sum_{1 \leq i' \leq \sigma, i' \neq i} f(i')/w$. **CountMin** can be used to estimate $\hat{f}_i$ with error at most ϵn with probability at least $1 - \delta$ using $\mathcal{O}(\frac{1}{\epsilon} \log \frac{1}{\delta})$ bits. The time per update is $\mathcal{O}(r)$ where $r = \log \frac{1}{\delta}$ and $w = \mathcal{O}(\frac{1}{\epsilon})$.

An example **CountMin** sketch is shown in Figure 7.1. When an item, i, is added to or removed from the sketch, one counter is incremented or decremented in each row of the **CountMin** sketch. The correct cell for each row is determined by the corresponding hash function. Formally:

$$\forall_{j<r} : count[j, h_j(i)] := count[j, h_j(i)] \pm 1$$

If the stream contains only positive frequencies, the frequency of item i can be estimated by returning the minimum count in the set.

$$\hat{a}_i = \min_j \ count[j, h_j(i)]$$

For streams allowing positive or negative frequencies, the frequency of i is estimated by returning the median of the r counts.

$$\hat{a}_i = \underset{j}{\text{median}}\ count[j, h_j(i)]$$

In general, the ϵ-bound of the **CountMin** sketch cannot be directly compared to the ϵ-bound of the **AMS** sketch (Cormode and Muthukrishnan, 2004). The **CountMin** sketch provides bounds relative to the L_1-norm, and **AMS** style sketches provide bounds relative to the L_2-norm. In practice, this is not a huge limitation, and both sketching approaches have been shown to be effective and efficient for skewed data collections (Charikar et al., 2002, Ganguly et al., 2004, Cormode and Muthukrishnan, 2005b, Cormode and Hadjieleftheriou, 2008).

Additional enhancements and applications of **CountMin** have been proposed in the literature. Conservative update, originally presented by Estan and Varghese (2002), is a heuristic method used to improve the frequency estimates produced by **CountMin** by minimizing collisions. It operates by only updating the minimum set of rows in the **CountMin** sketch. Using this approach, the update function for each row j for the event i becomes:

$$count[j, h_j(i)] := \max(count[j, h_j(i)], \min_{k<r}(count[k, h_k(i)] + 1))$$

7.3 Sketch Index Structure

Let $\mathcal{C}$ be a text collection partitioned into l documents $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_l\}$ containing at most σ_n unique terms. Here, a term t can be an n-gram, a sequence of n adjacent words. So, σ_1 represents the total number of unique 1-grams in the collection. An *inverted index*, $\mathcal{I}$ counts the number of times each term t appears in each document $\mathcal{D}_j$. Conceptually, this can be represented as a $\sigma_n \times l$ matrix, $\mathcal{M}$, as each indexed

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$	$\mathcal{D}_5$	$\cdots$
t_0	1	3	2	6	0	$\cdots$
t_1	2	0	3	1	2	$\cdots$
t_2	2	5	1	3	7	$\cdots$
$\vdots$						

Figure 7.2: Example matrix representation, $\mathcal{M}$, of a term-level, non-sparse inverted index, $\mathcal{I}$.

term t may appear in any document $mdoc_j$. Figure 7.2 shows an example of the matrix representation of an inverted index.

The following notation will apply for our discussion:

- $f_{d,t}$, the frequency of term t in document $\mathcal{D}_d$;
- $f_{q,t}$, the frequency of term t in the query;
- f_t, the number of documents containing one or more occurrences of term t;
- F_t, the number of occurrences of term t in the collection;
- l, the number of documents in the collection;
- σ_n, the number of indexed terms in the collection (vocabulary size); and
- $|\mathcal{C}| = \sum_{i=0}^{\sigma_1} F_{t_i}$, the total number of tokens in the collection.

In practice, $\mathcal{M}$ is sparse, and each row in $\mathcal{I}$ is often stored as a compressed posting list. An inverted index is a mapping of keys to a list of document counters. For each document identifier j, $f_{d,t}$ is maintained. Traditionally, each term in the vocabulary is stored explicitly in a lookup table.

Now, consider the case of constructing an inverted index of n-grams. The collection $\mathcal{C}$ can contain at most $(|\mathcal{C}| - n + 1)$ distinct n-grams. This number is often less than the σ_1^n possibilities, but still significantly larger than σ_1, thus increasing the number of potential rows in $\mathcal{M}$. Figure 7.3 shows an example of $\mathcal{I}$ when using n-gram terms.

$\vdots$				
$t_0\ t_1\ t_2$	$(\mathcal{D}_1, 1)$	$(\mathcal{D}_3, 3)$	$(\mathcal{D}_{10}, 2)$	$\cdots$
$t_0\ t_1\ t_3$	$(\mathcal{D}_2, 2)$	$(\mathcal{D}_3, 5)$	$(\mathcal{D}_4, 7)$	$\cdots$
$t_0\ t_1\ t_4$	$(\mathcal{D}_1, 1)$	$(\mathcal{D}_6, 5)$	$(\mathcal{D}_7, 1)$	$\cdots$
$\vdots$				

Figure 7.3: Example n-gram inverted index. For each n-gram, a list of documents and document frequencies are stored. Documents are sorted by identifier. If the n-gram is not present in a document, then the document is omitted from the index structure. Integer compression techniques can be used to reduce the total space requirements of the data structure.

We investigate how to apply the ideas presented by Cormode and Muthukrishnan (2005a) to fix the number of rows in $\mathcal{M}$ and still provide accurate statistical information. Interestingly, l is already static for a given collection, and $l \ll |\mathcal{C}|$. But, the number of rows, σ_n, increases with n, and we would like to minimize this overhead. Note that the total number of rows required in the sketch is proportional to $r \cdot f_t$. So, if we reduce $\mathcal{M}$ to a linear projection of f_t, we can use **CountMin** to accurately approximate $\hat{f}_t$. Recall that the expected number of collisions for t on any row in the sketch is $\sum_{1 \leq t' \leq \sigma, t' \neq t} f_{t'}/w$. Using a Markov inequality argument, Cormode and Muthukrishnan (2005a) show that by setting $w = 2/\epsilon$ and $r = \log 1/\delta$ in the sketch, the estimate $\hat{f}_t$ is at most $\epsilon \mathcal{F}_1$ with probability at least $1 - \delta$, where $\mathcal{F}_1$ is the first frequency moment $\sum_{1 \leq t' \leq \sigma_n} f_{t'}$, the sum of all of the frequencies. Note that this proof assumes that the hash functions selected are from a pair-wise independent family of hash functions.

In a sketch representation of an inverted index, each distinct term is replaced with a hash value where each hash value may represent more than one term. This reduction means that the vocabulary of n-grams no longer needs to be stored with the index. Figure 7.4 shows an example of our sketch-based indexing representation. If a simple hashing representation were used, then there is no mechanism available to

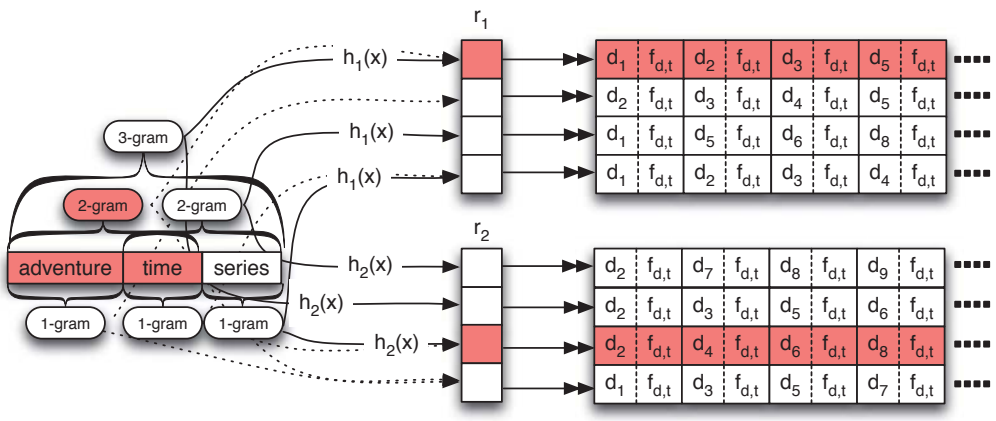


Figure 7.4: Example index representation of a sketch index data structure composed of two rows, $r \in \{0, 1\}$, each hash function is required to be pair-wise independent, and returns values in the range $[0, w - 1]$. Note that the postings lists in each row may contain hash collisions.

resolve collisions unless each term string is accessible to the table. However, using the collision mitigation strategy of a sketch, such as the method described for **CountMin**, we are able to reduce the probability that hashing collisions will result in incorrect results.

We note that it may be possible to use a perfect hash function to avoid collisions entirely for this type of index. However, one of the key advantages of a non-perfect hash function is that there is no requirement to retain the original vocabulary, thus avoiding unrealistic space requirements.

Our new indexing structure is composed of an $r \times w$ matrix of pointers to $r \times w$ postings lists. Conceptually, this matrix is equivalent to a **CountMin** sketch designed to estimate $\hat{f}_t$ with one twist: we do not simply use a single counter to aggregate $\hat{f}_t$, but rather allow multiple document counters attached in list-wise fashion to each cell in the **CountMin** sketch to form posting lists, as illustrated in Figure 7.4. These document counters are then used to aggregate $\hat{f}_{d,t}$.

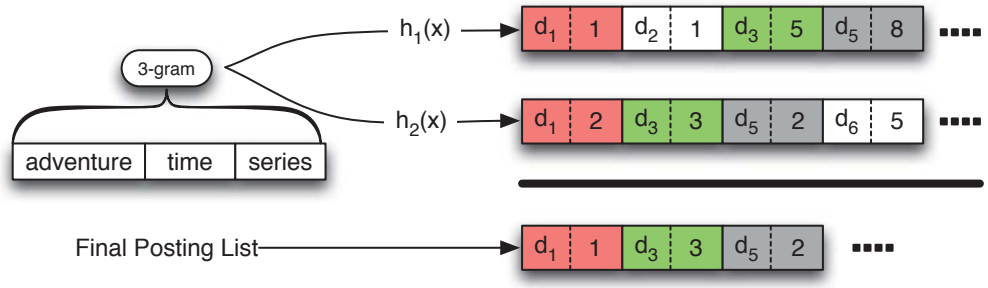


Figure 7.5: Example extraction of the statistics for a single term dependency in our sketch representation. The first two posting lists represent the 3-gram, ‘‘`adventure time series`’’, extracted from the sketch using the corresponding hash functions. The final posting list is simulated by intersecting r posting lists (in this case, $r = 2$). Colors identify matching documents for each $\hat{f}_{d,t}$ counter. The final posting list contains the minimum $\hat{f}_{d,t}$ from each matching document across r lists. For any document not represented in all r lists, the minimum is assumed as $\hat{f}_{d,t} = 0$.

This approach allows us to fix the size of the lookup table independent of the order of the n-grams being indexed. We do not attempt to fix the number of $\hat{f}_{d,t}$ counters. As in a standard inverted index, every term could appear in every document, producing a maximum of $l \cdot |\sigma_n|$ counters in the worst case. But, in practice, the distribution is skewed, and many terms have few non-zero $\hat{f}_{d,t}$ counters. Note that since the width of $|\sigma_n|$ is fixed in our approach, the number of counters is largely independent of the order of n, but rather some percentage of the counters are redistributed in the redundant postings lists.

We now discuss how to estimate the frequency of a particular n-gram using our approach. By using the biased estimation of a **CountMin** sketch of only positive counts, our estimates of $\hat{f}_t$, and subsequently $\hat{f}_{d,t}$, are guaranteed to be an overestimate of the true term counts. Furthermore, the same formal arguments using the Markov inequality and Chernoff bounds can be made for bounding $\hat{f}_t$, and subsequently $\hat{f}_{d,t}$, we could reasonably expect for each cell. So, to estimate $\hat{f}_t$ using **CountMin**, we would take $\min_j \ count[j, h_j(x_i)]$. But, each counter $count[\ ]$ is actually a pointer to a

postings list, containing approximately $\hat{f}_t$ counters. When the posting list for any t is requested, r posting lists are extracted and the *intersection* of the lists represents the $\min_j$ of the r postings lists. Figure 7.5 shows an example of the intersection process that represents the $\min_j$ for a given t.

This index is only intended to store and return collection- and document-level statistics, and to enable the ranked retrieval of documents for an input query. The use of hashing functions in the structure prohibit tasks that require vocabulary exploration. The proposed structure also cannot store positional data for n-grams, prohibiting the use of the structure in common post-retrieval tasks such as snippet generation. It is also possible to augment sketch-indexes to store positional data, but we do not explore this alternative here. The sketching techniques discussed here can be directly applied in a frequent index (Huston et al., 2011) and in term-pair indexes as proposed by Broschart and Schenkel (2012). Currently, the vocabulary of the sketch index and single-hash index are stored in memory with computed hash values used to index an array of file pointers. It is also possible to store this table implicitly, by using a B+Tree to map each hash value to the posting list data. This approach permits fine grained control of the memory requirements of the structure, but may introduce additional disk-seeks when executing queries.

7.4 Index Construction Algorithm

Construction of an sketch index is similar to the construction of a full index of term dependencies (see Chapter 5). A minor modifications are required to adapt this algorithm to construct the sketch index structure. The Sort-Based Sketch-Inversion algorithm details a modified Disk-Based One-Pass algorithm that constructs the sketch index structure.

In the simplest case, to generate a sketch index, a linear pass over the text collection with a sliding window of size n is performed. Each of the r hash functions

Algorithm SORT-BASED SKETCH-INVERSION

```
1: for each D_i ∈ C do
2:     for each term dependency, t_j ∈ (D_d) do
3:         for k ← [1 ... r] do
4:             append (k, h_k(t_j), D_d, f_{d,t}) to the output file O
5:         end for
6:     end for
7: end for
8: sort O lexicographically
9: write the data from O to sketch index structure I
```

are applied to each n-gram extracted to generate a set of r *term-ids*. The *term-id* data for each row is sorted. Then the algorithm writes the sorted data directly to the sketch index structure $\mathcal{I}_i$.

The cost of constructing our term dependency estimator is therefore equivalent to the cost of constructing an inverted index of n-grams with r repetitions. Specifically, we can bound the cost of constructing a sketch index to $\mathcal{O}(r \cdot |C| \log |C|)$, as the set of term postings must be sorted r times, once for each row.

As discussed in Chapter 5, it is relatively simple to distribute this indexing algorithm across a parallel computing environment. Again, the approach is to distribute documents across processors, to construct local sketch index shards on each processing node. Note that the r hash functions should be shared across the set of processors.

An important distinction from the frequent index, discussed in Chapter 6, the sketch index is amenable to dynamic index construction algorithms. A fully dynamic version of the index can be constructed by applying the dynamic indexing algorithms described by Büttcher et al. (2010). One approach is to maintain an in-memory version of the sketch index as documents are added to the collection. Periodically the in-memory sketch index is written to disk as an index shard. Index shards on disk are periodically merged to control the total number of index shards. In order to provide retrieval over the entire collection at any time, the posting list data extracted from the memory-shard and each disk index shard are merged at query time.

7.5 Experiments

Evaluation will consist of five empirical investigations. These investigations will focus on the observed error in point estimations, the retrieval effectiveness, the disk and memory space requirements of the structure, and the retrieval efficiency. Observations will be compared with previous discussed approaches. These investigations will test this structure's scalability, efficiency and test for any change in retrieval effectiveness.

We investigate the performance trade-offs of our new index structure using three TREC collections: Robust-04, GOV2 and ClueWeb-B. Both collections are previously presented in Chapter 3. The distributions, and vocabulary growth rates of different n-grams for each of these collections is shown in Chapter 5. In each of our experiments, we measure index properties and retrieval performance on n-gram data. Recall that an n-gram, is an ordered window of width 1, containing n terms, both are defined as any sequence of n sequential words. Recall from Chapter 5, as n increases, the vocabulary size increases dramatically. Secondarily, increasing n affects the skew of the statistical distribution of terms in a collection.

We compare the performance of our statistical n-gram estimator with four previously proposed index structures capable of storing and returning document-level statistics of n-gram term dependencies. We compare our approach with positional indexes, full indexes of n-grams, frequent indexes, query-log-based indexes and next-word indexes. We also compare the sketch index to a single-hashed index.

To ensure a fair comparison, all baseline index structures are implemented using the same set of modern index compression techniques, including d-gap and *vbyte* integer compression for posting list data, and prefix-based vocabulary compression for b-tree blocks. We use 32 kB b-tree blocks. We discuss these techniques for index compression in Chapter 2.

We compare the sketch index to 6 other index structures that provide the same n-gram statistics. The positional index and full index were previously defined and investigated in Chapter 5. The frequent index was defined in Chapter 6. We introduce the next-work index, and a query-log-based index structure here.

Next-word indexes, originally proposed by Williams and Zobel (1999), store a positional mapping for every word pair. The structure is divided into two files, a lexicon file and a vector file. The vector file stores positional posting lists for every word pair found in the collection. The lexicon file stores mappings from each word to a list of next-words and vector file offsets, and the vector file stores all posting lists. The auxiliary data structures in the next-word index can *only* be used for n-gram or phrase queries.

The query-log index can be considered an alternative method of selecting a subset of n-grams to index, based on a simulated query log. Using queries extracted from the AOL query log, we build an index of recently queried n-grams. See Chapter 3 for more details on the AOL query log. This method is analogous to the construction of a cache of intersected posting lists Ozcan et al. (2012). After ordering the query log by timestamps, we index all n-grams extracted from the first 90% of the AOL query log. The remaining 10% is reserved to test the retrieval efficiency of each of the index structures. Note that n-grams are extracted from each query, as if to be executed using the n-gram retrieval models presented in Section 7.5.2.

As a final sanity check and baseline, a singly-hashed index is also used. The single-hash indexing structure is equivalent to a single row of the sketch index. Each indexed n-gram is hashed to a b bit integer value. An index is constructed by associating each hash value with a posting list. This structure is implemented using a hash-indexed array of offsets into a file containing all posting lists. Recall that the likelihood of collisions decreases exponentially with respect to the number of independent hash functions used in any sketch-based data structure. Therefore, there is an implicit

trade-off in the number of hashes used and the bounded error rate. Our approach works with a single hash or many hashes in a similar manner.

Our experiments focus on four key aspects of our new statistical term dependency estimator: relative statistical error, retrieval effectiveness, disk and memory space requirements, and retrieval efficiency. A key component of this study is the investigation of the relationships between retrieval effectiveness, space requirements and retrieval efficiency for the sketch index, in comparison to each of the baseline data structures. We show that the sketch index provides valuable new trade-offs between efficiency, effectiveness and space requirements.

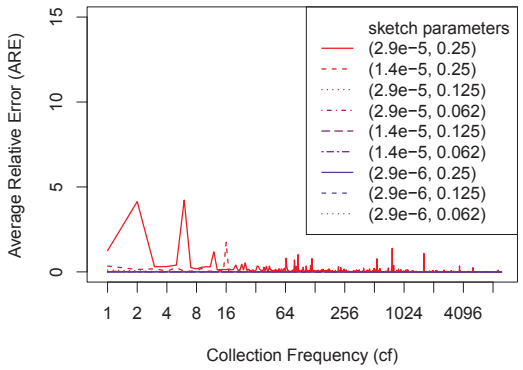
In each experiment, ϵ and δ are reported for each sketch index. These parameters determine the width and depth of the sketch used in the sketch index. The depth of the sketch is determined as $\lceil \log \frac{1}{\delta} \rceil$. In these experiment, we focus on 1, 2, and 3 row sketch indexes, specified by $\delta \in \{0.5, 0.25, 0.125, 0.0625\}$, respectively. The width of the sketch is determined as $w = \frac{2}{\epsilon}$. So, where $\epsilon = 2.9e-06$, the width of the sketch is 554751 cells. The width of the hash values required from the hash function, for this value of ϵ, is at least $\lceil \log(554751) \rceil = 20$ bits.

Each index structure we investigate in this section is implemented as an extension to the Galago package, provided by the Lemur Toolkit [1]. All timing experiments were run on a machine with 8-core Intel Xeon processors, with 16 GB of RAM, running the CentOS distribution of Linux, using a distributed, network-attached, 4-node Lustre file system to store index data. We measure the CPU time taken for at least 10 consecutive runs, and report the average in each experiment.

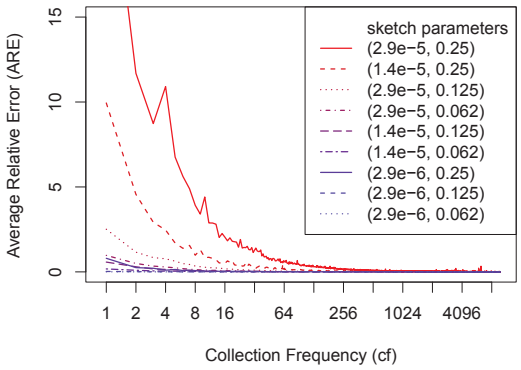
7.5.1 Estimation of Collection Frequency

As discussed previously, sketch indexes provide an attractive trade-off between space usage and accuracy. In this section, we investigate the relationship between the

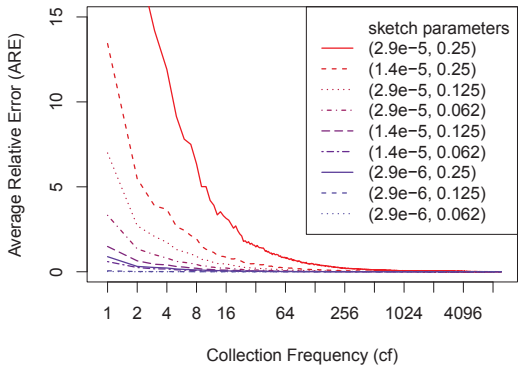
[1] A component of The Lemur Project, `http://www.lemurproject.org/galago.php`



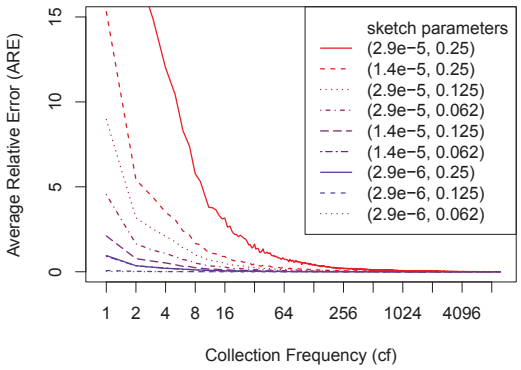
(a) 1-gram sketch index



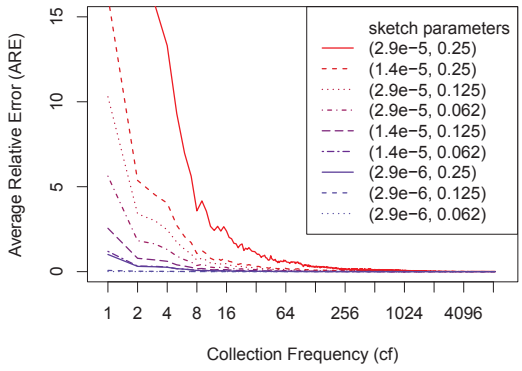
(b) 2-gram sketch index



(c) 3-gram sketch index



(d) 4-gram sketch index



(e) 5-gram sketch index

Figure 7.6: Average relative error of n-gram frequency statistics extracted from 10 instances of sketch indexes over Robust-04 data, using each set of parameters. Sketch index parameters, (ϵ, δ), shown are $\epsilon \in \{2.9 \cdot 10^{-5}, 1.4 \cdot 10^{-5}, 2.9 \cdot 10^{-6}\}$, and $\delta \in \{0.25, 0.125, 0.062\}$. Note

(ϵ, δ) parameters and the quality of approximation by comparing the relative error of our approach to the true collection statistics. We show results computed on indexes created over n-grams, extracted from the TREC Robust-04 collection. The Average Relative Error (ARE) is defined as the average of the absolute difference between the true value and the estimated value. In this case:

$$ARE_n = \frac{1}{|\sigma_n|} \sum_{t \in \mathcal{T}_n} \frac{|F_t - \hat{F}_t|}{F_t},$$

where $\mathcal{T}_n$ is the set of all unique terms (n-grams) of size n and $\sigma_n = |\mathcal{T}|$.

Figure 7.6 shows ARE_n values grouped by F_t for several different n-grams using our approach with a variety of parameters. Data shown in this graph is aggregated from 10 instances of sketch indexes with each parameter setting. The x and y axes are identical for each graph, allowing direct comparison.

First, this data shows that conservative settings for (ϵ, δ) can ensure a low error rate in the estimation of collection statistics. Additionally, we can see that using overly restrictive values of ϵ and δ can degrade our estimates, particularly for infrequent items. This insight is not surprising, since summary sketching has primarily be applied in networking scenarios that require only the top-k items in a set to be accurately estimated. This problem is referred to as the "heavy-hitter" problem in the literature. Nevertheless, accurate estimates are possible using these approaches if conservative ϵ and δ values are used.

The graphs also show the relationship between error rate and the skew of the indexed data. Recall from Chapter 5, increasing the number of terms, n, decreases the skew of the term frequency in the collection. Figure 7.6 shows that as the skew of indexed data decreases, ϵ must also be reduced in order to ensure a comparable relative error.

In particular, we see that $\epsilon \leq 2.9 \cdot 10^{-6}$ and $\delta \leq 0.062$ produce a low relative error for all n-grams tested in the Robust-04 collection. Relative error for other important

statistics including document count and document frequency for each n-gram were also evaluated. The graphs showing error rates for these statistics are omitted from this study as the overall trends remain the same.

We now relate these observations to the theoretical bounds discussed previously. Recall that the expected error rate is controlled by ϵ and δ. Taking an example from our above graphs, we set $\epsilon = 2.9 \cdot 10^{-6}$, and $\delta = 0.25$. Theoretically the expected error for this type of sketch is $\epsilon \cdot |C| \leq 735$, for the Robust-04 Collection. We expect to see that no more than 25% of estimated collection frequencies overestimate the true statistic by more than 735. In practice, we obtain a much smaller observed error rate. In the data collected and summarized in the ARE graphs for these parameters, no n-gram collection frequency values are overestimated by 735. The highest overestimation of the collection frequency of an n-gram, observed in this data is 261. From this data, it's clear our observations do not contradict the probabilistic error bounds. The difference between theory and practice here is that the theory assumes that each sketch cell, in this case a posting list, just stores a single collection frequency value. By intersecting the stored posting lists, we obtain a more conservative observed error.

7.5.2 Retrieval Effectiveness

We now investigate the effect of the use of sketch indexes on information retrieval effectiveness. We focus on testing that sketch indexes can be used to store and retrieve n-gram features for information retrieval without degrading retrieval effectiveness. We seek to investigate if this data structure introduces a risk of compromised retrieval effectiveness. For comparison, the relationship between hash-table size and retrieval effectiveness for the single-hashed index is also evaluated in these experiments. All the other benchmark index structures provide accurate collection statistics, thus they are not specifically evaluated in this section.

Given that the indexes we are focusing on in this chapter store n-gram statistics, it is appropriate to an n-gram based retrieval model. We use the strongest performing n-gram retrieval model studied in Chapter 4, Uni+O234. This model ranks documents using 1- to 4-grams. In Chapter 4, we do not observe any statistical differences between this model and the sequential dependence model (SDM) (Metzler and Croft, 2005), for each of the tested collections.

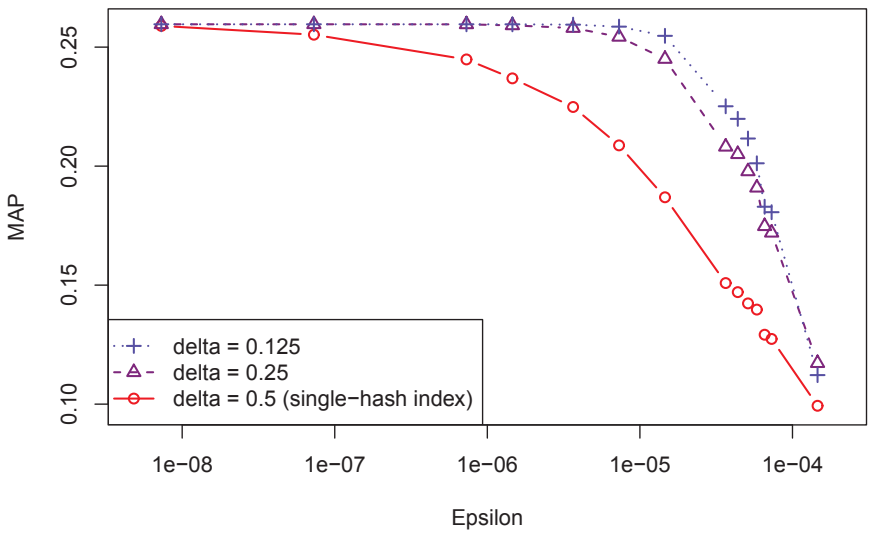
The Uni+O234 retrieval model is defined as:

$$\begin{aligned} P_{Uni+O234}(D|Q) \stackrel{rank}{=} & \sum_{i \leq |Q|} \lambda_T \log P(q_i|D) \\ & + \sum_{i \leq |Q|-1} \lambda_{O2} \log P(\#od1(q_i, q_{i+1})|D) \\ & + \sum_{i \leq |Q|-2} \lambda_{O3} \log P(\#od1(q_i, q_{i+1}, q_{i+2})|D) \\ & + \sum_{i \leq |Q|-3} \lambda_{O4} \log P(\#od1(q_i, q_{i+1}, q_{i+2}, q_{i+3})|D), \end{aligned}$$

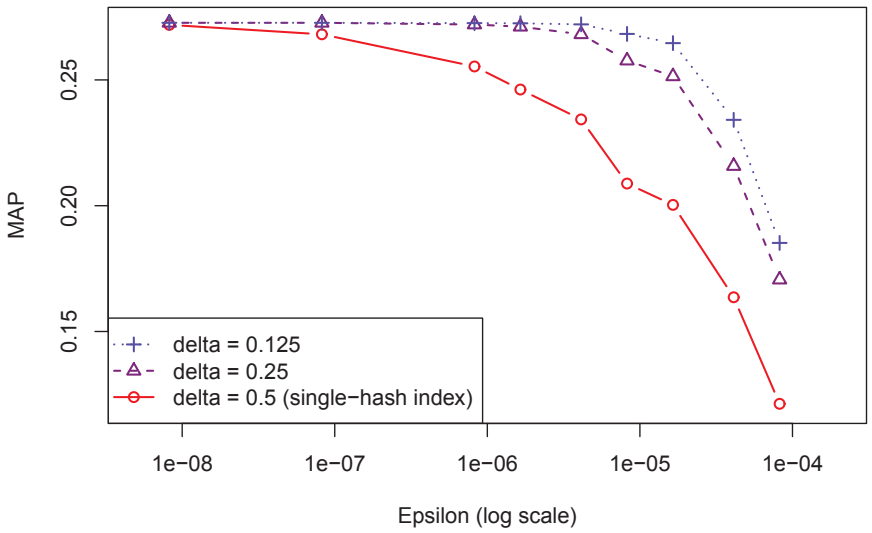
where the model is parameterized using 4 weights ($\Lambda = \lambda_T, \lambda_{O2}, \lambda_{O3}, \lambda_{O4}$), and q_i is the i^{th} term in query Q. Recall that the $\#od1$ operator, originally defined by Metzler and Croft (2005), is an ordered window operator, it matches instances of n-grams in each scored document, and returns the number of matches found, where n is as the number of terms provided to the operator.

We now investigate the effect n-gram sketch indexes have on the effectiveness of the Uni+O234 retrieval model. We explore the relationship between different sketch parameters and retrieval performance (MAP). Sketch indexes used in this experiment each contain statistics for all n-grams ($1 \leq n \leq 4$) in the collection. Other retrieval metrics (nDCG@20 and P@20) were also evaluated, and similar trends were observed.

Figure 7.7 shows that retrieval performance (MAP) degrades as the ϵ parameter is increased and the sketch table width decreases correspondingly. We can see that for each value of δ, there is a threshold value of ϵ, below which retrieval effectiveness is



(a) Uni+O234 model, Robust-04



(b) Uni+O234 model, GOV2

Figure 7.7: Retrieval effectiveness using sketch indexes measured using MAP, varying the (ϵ, δ) parameters, for each collection. Note that the sketch index where $\delta = 0.5$ is equivalent to a single-hash index, $\lceil \log(1/0.5) \rceil = 1$. Each data point is the average retrieval performance from 10 sketch index instances.

Table 7.1: Sketch parameters and the corresponding sketch table sizes. Observed retrieval effectiveness (MAP) for each of the parameter settings in this table is within 1% of observed retrieval effectiveness for the Uni+O234 retrieval model, using oracle-tuned parameters.

Collection	Delta	Epsilon	Sketch Width	Sketch Depth
Robust-04	Single-hashed	$7.3 \cdot 10^{-8}$	$27,457,393$	1
Robust-04	0.25	$3.6 \cdot 10^{-6}$	$554,752$	2
Robust-04	0.125	$1.4 \cdot 10^{-5}$	$143,067$	3
GOV2	Single-hashed	$8.1 \cdot 10^{-9}$	$247,116,529$	1
GOV2	0.25	$1.6 \cdot 10^{-6}$	$1,235,582$	2
GOV2	0.125	$8.1 \cdot 10^{-6}$	$247,116$	3

identical to the oracle. A summary of these threshold parameter settings is shown in Table 7.1. It is important to observe that the sketch parameters only need to grow sub-linearly in the size of the collection. GOV2 is almost 1,000 times longer than Robust-04 (as measured by collection length), and we observe that sketch parameters grow only sub-linearly, even while ensuring no change in retrieval effectiveness. Sketches of 3 rows are observed not to dramatically change the threshold settings. As such, we focus on the 2 row sketch in the following experiments.

7.5.3 Memory and Disk Space Requirements

In this section memory and disk space requirements are evaluated for sketch indexes over a variety of sketch parameters and across different collections. We then compare disk and memory requirements to benchmark index structures.

Figure 7.8 shows memory requirements for a range of sketch parameters. Figure 7.9 shows the specific memory requirements of sketch indexes that maintain accurate retrieval effectiveness for the Robust-04 and GOV2 collections (see Table 7.1). We observe that the memory requirements for sketch indexes grows much slower than the single hash index structure. If each sketch cell stores an 8 Byte file offset for the postings data, the memory requirement for a sketch index of all 1-to-4-grams in the GOV2 collection is just 19 MB. Over the same collection, using similar 8 Byte offset

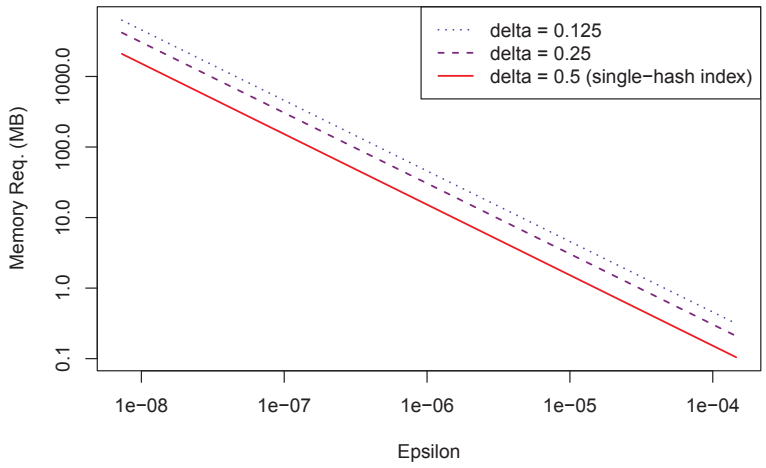


Figure 7.8: Memory requirements for a range of sketch index parameters. Note both x and y axes are in log scale.

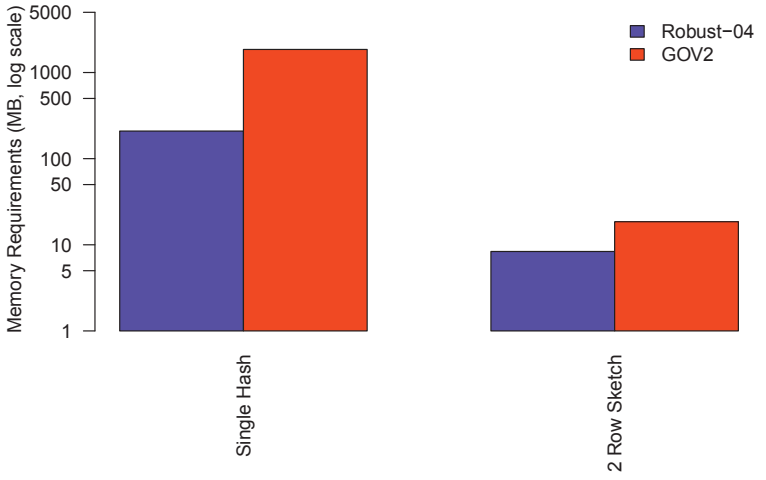


Figure 7.9: Memory requirements for single hash structure vs a 2 row sketch index, using parameters specified in Table 7.1. This graph assumes that each cell in the hash tables is an 8 Byte file offset to the posting list data. Note y axis is in log scale.

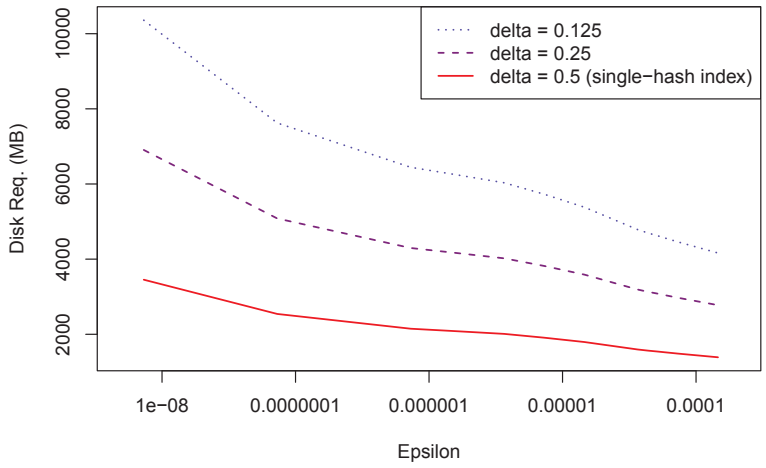


Figure 7.10: Disk requirements for sketch indexes of the Robust-04 collection. Note the x axis is in log scale.

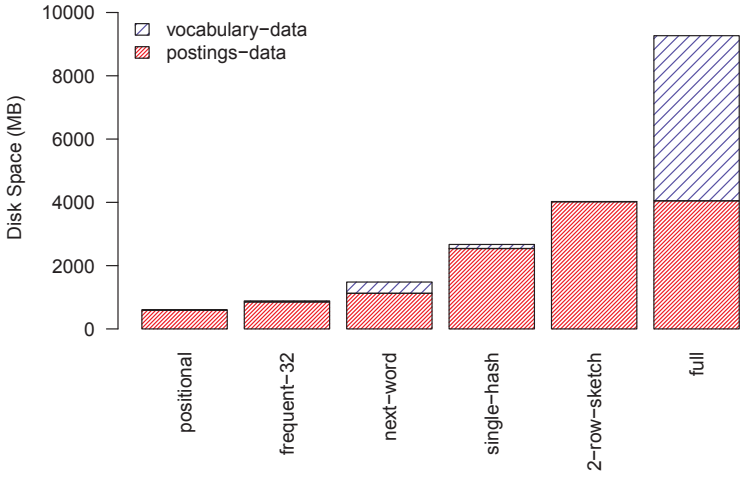


Figure 7.11: Disk requirements for sketch indexes of the Robust-04 collection. Sketch index parameters are selected such that retrieval effectiveness is not compromised (see Table 7.1).

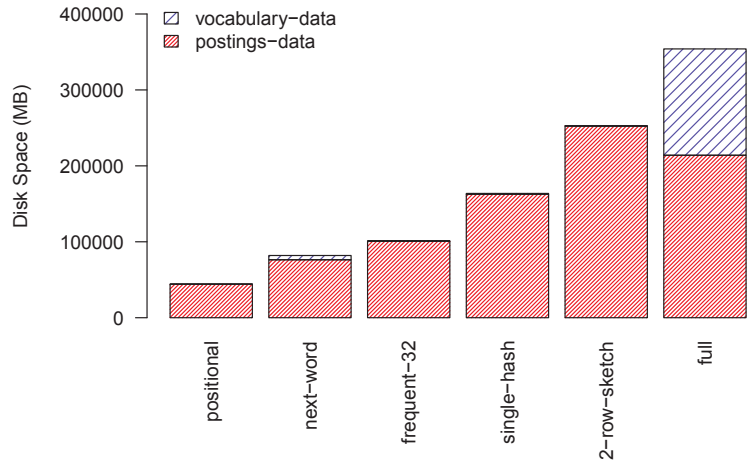


Figure 7.12: Disk requirements for sketch indexes of the GOV2 collection. Sketch index parameters are selected such that retrieval effectiveness is not compromised (see Table 7.1).

values, the single-hashed index requires almost 2 GB of RAM. This data shows that the sketch index is able to scale to large collections without introducing unreasonable memory requirements.

Figure 7.10 shows the disk requirements of sketch indexes on the Robust-04 collection for a range of sketch parameters. This data clearly shows that naïve use of sketch indexes can result in inefficient use of disk resources. Figures 7.11 and 7.12 show disk requirements for sketch indexes compared to the disk requirements of the baseline index structures for the Robust-04 and GOV2 collections. This data shows that the sketch index uses significantly less space than the full index, but more than each of the other baseline methods. The vocabulary data for a 2-row sketch index is less than 0.01% of the disk requirements of the vocabulary data for a full index of 1-to-4-grams extracted from the GOV2 collection. However, the postings data stored in the same sketch index grows by a factor of 1.2 over the postings data stored in the full index. This implies that the sketch index is most effective when storing the

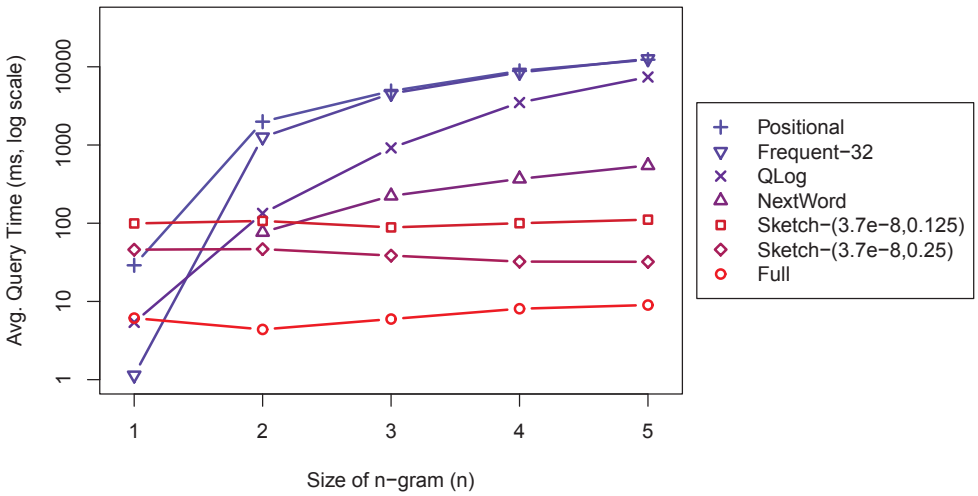


Figure 7.13: Query processing times for indexes over the ClueWeb-B collection. Each data point is the average of 10 runs of 10,000 phrase queries.

vocabulary data is a large cost of the inverted index, as in an inverted index of term dependencies.

7.5.4 Retrieval Efficiency

We now evaluate the retrieval efficiency of our statistical estimator relative to the other index structures for n-gram queries. For this experiment, we sample queries from the AOL query log. The AOL query log consists of over 20 million unique web queries that users submitted to the AOL search engine in 2006. In these experiments, we execute queries on indexes of the ClueWeb-B collection. By using a web collection to target the queries, no query translation methods Webber and Moffat (2005) are required. We omit the single-hash index from this experiment, as it operates identically to the full index structure.

The first 90% of the time-ordered query log is used to create the query log cache index structure. Test queries are sampled from the remaining 10% of the log. From this subset of the query log, we uniformly at random sample $10,000$ n-grams ex-

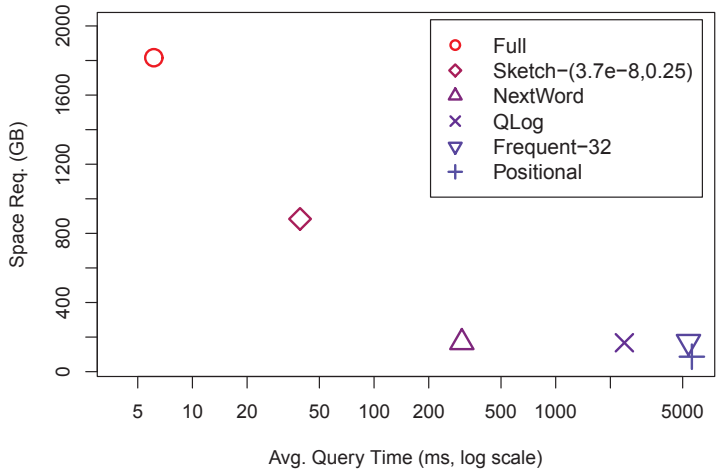


Figure 7.14: Query processing time versus space usage. Query processing time as a function of index space requirements. Each data point is the average of 10 runs of 10,000 of n-gram queries. The size of the query for each data point is indicated on the graph.

tracted from queries for each size, $n \in \{1, 2, 3, 4, 5\}$. The n-grams extracted from this query log represent a random sample of query features that are required to be computed for the n-gram based retrieval model used in Section 7.5.2. We note that this sampling technique would also be appropriate for several other types of retrieval models. Several linguistic and machine learned techniques segment or classify lists of terms extracted from a query into potentially valuable phrases, windows, n-grams, and other dependent sets of terms (Bendersky and Croft, 2008, Bergsma and Wang, 2007, Shi and Nie, 2010). These retrieval models each require collection statistics for all candidate n-grams or candidate query term sets. Therefore, each of these models requires that index structures provide access to statistics for any n-gram that may be queried, including the ability to efficiently determine if the n-gram occurs in the collection.

The processing speed for each index structure over a each size of n-gram is measured as the average of 5 timed runs of the corresponding sample of queries. To

ensure the order of extracted n-grams does not affect the results, the order of each run is randomized. The retrieval system is initialized for each experiment by running a randomly selected sub-sample of 2,000 queries. This process ensures that a portion of the index data is held in memory-based file buffers, as it would be in a live retrieval system.

Figure 7.13 shows query processing time as the length of the query increases for each index structure. Note that times shown in this graph are displayed in log scale. All data points in the graph are significantly different from each of the other index structures, $\alpha = 0.05$ for all pairs using the Fisher randomization test. This graph shows that the query processing time of the sketch index data structure is significantly faster than the positional, frequent, query-log and next-word indexes. Unlike position-based indexes, we can see that the sketch index is scalable in the length of the n-gram, since as n increases, the time to process n-gram queries does not increase with n.

The full index processes an average 4-gram in around 10 ms, while the sketch index processes the same average 4-gram in 33 ms. The other index structures (positional, frequent and next-word indexes) all process longer n-grams between 1 to 3 orders of magnitude slower on average than the sketch index structure. In particular, sketch indexes can be over 400 times faster than a positional index, and 15 times faster than next-word indexes, for processing 5-gram queries.

Figure 7.14 shows the trade-off between query processing speed and space usage. Data shown is the total space requirements to process 1-to-5-gram queries for each index structure, and the average time to process all of the sampled queries used in the timed experiments above. Query processing times are shown in log scale. Data structures that provide the best trade-off will approach the origin - that is the query processing time, and space requirements are minimized. This graph clearly demonstrates that our n-gram statistical estimator offers a new and effective trade-off between space usage and query efficiency compared with all other baselines.

7.6 Summary

In this chapter, we investigated the problem of accurately estimating document and collection level term dependency statistics in large data collections. Existing solutions for this problem either require large amounts of disk space, or are inefficient for query processing in practice. We have presented a novel approach to estimating n-gram statistics for information retrieval tasks. By using frequency sketching techniques developed for data streaming applications, we can accurately estimate collection and document level statistics, and provide an attractive trade-off between space and relative error. Furthermore, we show how to bound the space usage of the data structure. Importantly, the number of distinct n-grams stored in the sketch is logarithmically linked to the size of the sketch, allowing us to scale up to very large collections, as empirically observed in experiments focusing on memory space requirements.

We demonstrated that our approach is efficient in both time and space requirements, and can provide a low error rate for all of the statistics being estimated. Empirically, we have shown that the sketch index can reduce the space requirements of the vocabulary component of an index of 1-to-4-grams extracted from the GOV2 collection to less than 0.01% of the requirements of an equivalent full index. We have shown that sketch indexes can process queries considerably faster than both positional indexes, and next-word indexes. Unlike frequent indexes and query-log cache approaches, our method does not require an auxiliary index to calculate statistics for unseen or infrequent n-grams. This new index representation provides an attractive alternative to other state-of-the-art approaches depending on n-gram statistics for retrieval tasks.

The sketch index data structure provides a new and useful trade-off between query processing time and space requirements for n-gram queries. Importantly, we also have shown that this index structure is scalable in both query processing time and space

requirements for the size of the queried n-gram, n, and for the size of the collection, $|C|$.

Finally, the index structures described encourage the exploration of n-grams as query features. We have initiated this exploration through a simple n-gram based retrieval model in this study. Our retrieval model can be executed efficiently using sketch index structures. We have empirically shown these models to be significantly more effective than the query likelihood retrieval model, and not perform significantly differently than the sequential dependence model. Furthermore, using the sketch index data structure, n-gram retrieval models can be executed more efficiently than positional index-based implementations of the sequential dependence model.

Experiments performed in this chapter focus on n-gram data. We intend to extend this analysis to investigate the application of this structure to unordered windows. We can make some predictions about this application using data gathered in Chapter 5. This data shows that the number of postings grows considerably with the width of the unordered window, and that the size of the vocabulary grows at approximately the same rate as n-grams. Further, this ratio between vocabulary and posting list space requirements is similar to the ratio observed for n-grams. This observation, combined with the observations made for n-grams above, implies that collection and document statistics of all unordered windows containing 3 or more terms would be efficiently stored in this structure.

The sketch index primarily reduces the space requirements of vocabulary data. The stored postings data is unchanged. Multi-dimensional sketching techniques, such as those presented by Thaper et al. (2002), may be an effective method of sketching posting list data. A problem for this technique is that the posting lists are used in query processing algorithms to determine which documents contain at least one of the query terms. Sketching techniques would eliminate this efficiency improvement. Another alternative is to use a secondary CountMin sketch to approximate the data

stored in each posting list. This option requires that all data in each required posting list is read into memory for the execution of a query. If this approach is used, we can consider performing matrix-like operations to simultaneously combine document statistics for each query term and term dependency, and thereby, score each document in the collection. However, a final iteration over the documents is still required to collect and return the top k documents.

CHAPTER 8

RETRIEVAL OPTIMIZATION

8.1 Introduction

In Chapter 4, we extensively investigated a range of proximity-based dependency models. We identified SDM and WSDM-Internal to be the strongest performing dependency models. In Chapters 5, 6, and 7, we presented and analyzed index structures that support the storage and retrieval of ordered and unordered window statistics. In this chapter, we combine these lines of research and investigate the execution of SDM and WSDM-Internal, using each of the term dependency indexes analyzed in this thesis.

Recall from Chapter 2, that a query processing algorithm, sometimes called a query evaluation strategy, is defined as a process that extracts the required statistics stored in a set of index structures to determine the k documents that are most likely to be relevant to a query, for a given retrieval model. Any viable algorithm must be able to return these k documents in an extremely short amount of time.

We can consider a retrieval model to be a function that combines a set of collection-level and document-level statistics, stored in the index, and returns a probability, or score, for each document in the collection. There are two types of methods of reducing the query processing costs: reducing the number of documents scored, and reducing the computation required to demonstrate that a particular document is not in the top k for the collection. In general, optimizations can be classified as 'safe', 'rank-k-safe', or 'unsafe' (Turtle and Flood, 1995).

As discussed in Chapter 2, there are two classes of query processing algorithms: term-at-a-time and document-at-a-time. Term-at-a-time algorithms have been shown to be effective for frequency- or impact-ordered indexes (Anh and Moffat, 2006, Strohman and Croft, 2007). However, these algorithms have large memory requirements, reducing the amount of memory available for caching posting lists and top-k result lists. Document-at-a-time algorithms only require memory sufficient to store the current top k documents, and the data required to store the next document. In order to be rank-safe, these algorithms cannot make any assumptions about the utility of unprocessed documents, and so, must completely process the posting lists of a subset of the query terms.

We restrict the scope of this chapter to two document-at-a-time algorithms, Document-at-a-Time and Max-Score. The Document-at-a-Time algorithm makes very few assumptions about the retrieval model being executed. Max-Score assumes that the retrieval model can be decomposed into the weighted sum of contributions, where each contribution can be bounded with a maximum, and minimum contribution.

The major contributions in this chapter include:

- analysis of the optimization of query processing models for dependency retrieval models; and
- empirical evaluation of the application of presented term dependency indexes for query processing algorithms.

8.2 Query Processing Algorithms

8.2.1 Document-at-a-Time

The Document-at-a-Time algorithm is a naïve algorithm that scores all documents in the collection to determine the top k documents. Documents are scored in increasing order of document identifier. In this algorithm, the retrieval model M is

Algorithm DOCUMENT-AT-A-TIME

```
1: initialize heap structure, H
2: retrieve document lengths/priors, L
3: for q_j ∈ Q do
4:     retrieve posting list iterator, P_{q_j}, for query term, q_j
5: end for
6: extract all collection statistics, C, from posting list iterators, P_Q
7: for d_i ∈ D do
8:     compute score, S = M(P_Q[d_i], L[d_i], C)
9:     if |H| < k or min(H) < S then
10:        insert pair, (d_i, S), into heap, H
11:        if |H| > k then
12:            remove min(H) from H
13:        end if
14:    end if
15: end for
16: return  return the k documents in H
```

used to score each document d_i for the input query, by combining statistics relevant to d_j, from the posting list iterators. A min-heap of size k is used to ensure that the top k documents are retained.

A simple, safe improvement to this algorithm is to score only documents that contain one or more of the query terms. This optimization can be implemented by modifying line 7 of the DOCUMENT-AT-A-TIME algorithm, to inspect the posting list iterators and determine the next candidate document. This optimization also requires a modification at line 14, to ensure that iterators are moved past the scored document, d_j.

The extraction of collection statistics at line 6 of the algorithm is vital for the computation of many different types of retrieval models. For example, the SDM requires the collection frequency of each query term, ordered window, and unordered window to score each feature. WSDM-Int further requires the document count of each query term, ordered window and unordered window to determine the weight

assigned to each retrieval feature. See Chapter 4 for the definition of each of the retrieval models.

If a positional index is used, each of these statistics must be collected by recombining positional data using the algorithms discussed in Chapter 5. This extra pass over the posting list data can be a major bottleneck in the execution of dependency retrieval models.

However, if a full, frequent or sketch index of the required ordered and unordered windows is available, then the collection statistics can be extracted directly from the prefix of the associated posting list. These types of indexes eliminate the extra pass over the postings data.

An important advantage of Document-at-a-Time algorithms over term-at-a-time algorithms, is a very low memory memory requirement. At any time during processing, only the top k documents, and all data required to score the current document, must be retained. Free memory can be used to cache frequently accessed portions of the index structures, reducing random-disk-access costs.

For the positional index structure, we can also consider caching the computed term dependency statistics between the first and second passes. This could save a large amount of processing time. However, for large collections, the memory requirements may render this optimization infeasible. In our experiments, we do not use this optimization.

8.2.2 Max-Score

Max-Score is an efficient rank-k-safe query processing algorithm (Turtle and Flood, 1995). This algorithm assumes that the retrieval model can be decomposed into the sum of a set of feature contributions. It requires an estimation of the maximum contribution of each retrieval feature prior to scoring documents. The algorithm is able to short circuit the evaluation of each document. After processing each re-

Algorithm DOCUMENT-AT-A-TIME

```
1: initialize heap structure, H
2: retrieve document lengths/priors, L
3: for q_j ∈ M(Q) do
4:     retrieve posting list iterator, P_{q_j}, for query term, q_j
5: end for
6: extract all required collection statistics, C, from posting list iterators, P_Q
7: for retrieval model feature, f_i ∈ F do
8:     estimate the maximum contribution c_i, using C
9: end for
10: compute the total maximum contribution T = Σ_i c_i
11: sort feature set F, by descending maximum contribution
12: while |H| < k do
13:     determine candidate document, d_j = min(P_Q)
14:     initialize score, S = C
15:     for f_i ∈ F do
16:         compute actual contribution, a_i, of feature f_i, for document d_j
17:         adjust score according to true contribution of feature f_i, S −= (c_i − a_i)
18:     end for
19:     insert pair, (d_j, S), into heap, H
20: end while
21: while not done iterating P_Q do
22:     determine candidate document, d_j = min(P_Q)
23:     initialize score, S = T
24:     for f_i ∈ F do
25:         compute actual contribution, a_i, of feature f_i, for document d_j
26:         adjust score according to true contribution of feature f_i, S −= (c_i − a_i)
27:         if S < min(H) then
28:             break
29:         end if
30:     end for
31:     if min(H) < S then
32:         insert pair, (d_i, S), into heap, H
33:         remove min(H) from H
34:     end if
35: end while
36: return return the k documents in H
```

trieval feature, the algorithm checks if the maximum contribution of all unevaluated features could result in a score higher than the minimum score in the heap, to decide

if the algorithms can move on to the next document, without computing the entire document score.

Turtle and Flood (1995) assert that the retrieval model features should be sorted into decreasing order of inverse document frequency. This ensures that the smallest posting lists are processed first. In TF-IDF scoring functions, this approach is very effective. In our implementation of the Max-Score algorithm, retrieval features are sorted by descending maximum contributions. This heuristic modification been found to improve the efficiency of the Max-Score algorithm for retrieval models that are based on the language modeling framework.

In addition to using collection statistics used to compute the scores for retrieval model features, certain collection statistics are required to place bounds on the contributions of each feature. For Dirichlet smoothed language model features, as used in QL, SDM and WSDM, the maximum document frequency, $max_d(tf_{d,t})$, is used to bound the maximum contribution. Similar to the Document-at-a-Time algorithm above, if positional indexes are used, the computation of this statistic for term dependency features will require the processing of all positional posting data.

Macdonald et al. (2011a) and Macdonald et al. (2011b) present methods of estimating the maximum contribution of term dependency features. These estimates can only be used if the retrieval model used does not also require collection statistics for term dependency features. Using estimated collection statistics in the evaluation of a retrieval model is not the focus of this chapter.

It is possible to improve the efficiency of the algorithm by keeping track of a pivot point in the list of retrieval features. The pivot point is defined as the first feature at which the sum of maximum contributions of the remaining features is larger than the minimum score in the heap. This pivot point can be used to avoid repeatedly checking if the running score is smaller than the minimum score on the heap, at line

27 in Max-Score. After each high scoring document is added to the heap, the pivot point must be recomputed.

8.3 Experiments

In these experiments, we seek to evaluate the efficiency of an end-to-end retrieval system that uses the proposed term dependency indexes. We compare the performance of five sets of index structures, using two large TREC collections, GOV2 and ClueWeb-09-Cat-B and two dependency retrieval models, SDM and WSDM-Int. Both collections were described in Chapter 3, and both of the retrieval model were defined in Chapter 4. Both of these retrieval models require term, ordered window (`od-w1-n2`), and unordered window (`uw-w8-n2`) document and collection statistics. We set the parameters in each model as the average of the parameters learned using 5-fold cross-validation, as reported in Chapter 4.

We use five different sets of index structures to provide access to document- and collection-level statistics for each retrieval feature in this experiment. Each of the sets of index structures is augmented with a list of document lengths, held in memory, and a disk-based B+Tree that maps document identifiers to document names.

The first index structure is the positional index. This index structure is a common index structure, implemented in several open source search engines, including Indri, Galago, Terrier, and Lucene. This is partly because it is a space efficient method of supporting the computation of a wide range of different retrieval models. We discuss this index structure in Chapter 5.

The second index structure is the full index. This index directly stores the statistics required by the retrieval model. A posting list is stored for each term, ordered window and unordered window in the target collection. We discuss this index structure in Chapter 5.

The third index structure is the frequent index, analyzed in Chapter 6. This index structure contains a subset of the vocabulary of the full index. Statistics for terms, ordered windows and unordered windows are stored where the collection frequency of each item is above the set threshold. We set the threshold to 10 for each of these experiments. In this experiment, we use this index structure in a lossy manner, as discussed in Section 6.3.

The fourth index structure is the frequent index, augmented with a full index of terms. The difference is that statistics for all terms are retained. Again, we use this index structure in a lossy manner.

Finally, the sketch index stores an (ϵ, δ)-approximation of the statistics stored in the full index. In these experiments, we set $\epsilon = 6.06 \cdot 10^{-8}$ and $\delta = 0.25$ for the GOV2 collection, and, $\epsilon = 5.53 \cdot 10^{-8}$ and $\delta = 0.25$ for the Clueweb-09-Cat-B collection. These settings were selected to be conservative, ensuring that the retrieval effectiveness is unaffected by the approximate nature of the structure.

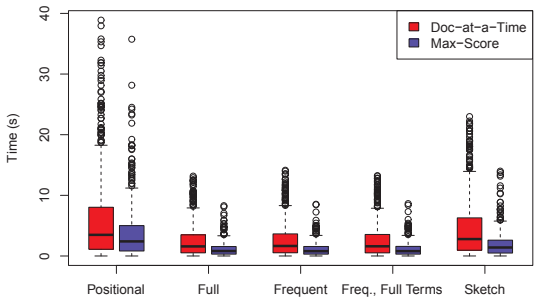
Each of these index structures is implemented in the Galago search engine. [1] Each timing experiment is executed on one of 4 identical machines. Each machine has access to 48 GB of RAM, and 2, 6-core Intel Xeon processors.

8.3.1 Query Processing

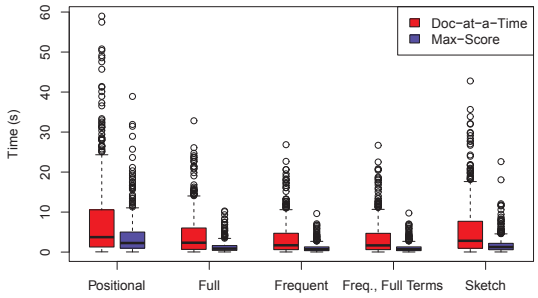
To investigate the retrieval efficiency of each of these structures, we sample queries from the TREC Million query tracks. For details of these data sets see Chapter 3.

We sample 500 short queries, and 500 long queries from the topics used in 2007, 2008 TREC Million Query Tracks, for the GOV2 collection, and a further 500 short queries, and 500 long queries from the 2009 TREC Million Query Track for the ClueWeb-B collection. A short query is defined as a query consisting of between 2 and 3 terms, and a long query is defined as a query consisting of between 4 and

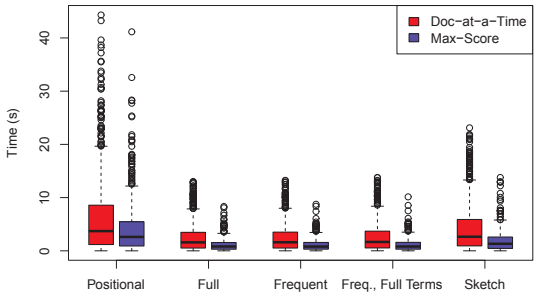
[1] A component of (The Lemur Project), `http://www.lemurproject.org/galago.php`



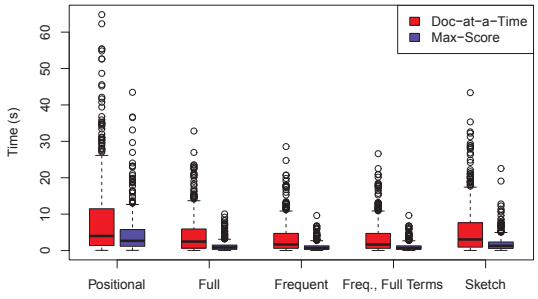
(a) GOV2, Short queries, SDM



(b) ClueWeb-09-B, Short queries, SDM

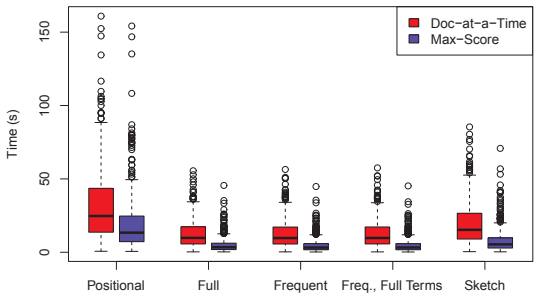


(c) GOV2, Short queries, WSDM-Int

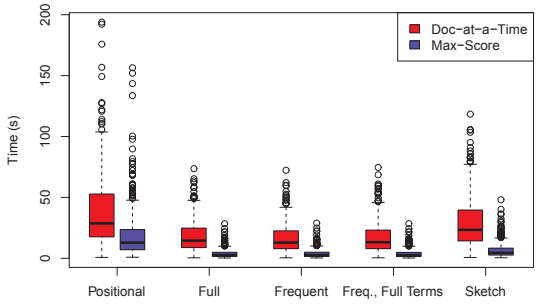


(d) ClueWeb-09-B, Short queries, WSDM-Int

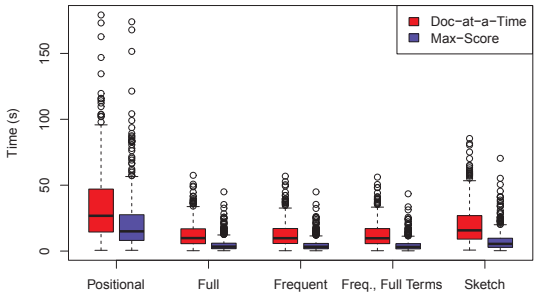
Figure 8.1: Average query execution times for short queries (2 to 3 terms). On each graph the query processing times are measured for both the Document-at-a-Time and Max-Score algorithms. Graphs shown span two collections (GOV2 and ClueWeb-09-B), two models (SDM, and WSDM-Int). Each query processing time is the average of 5 repeated executions.



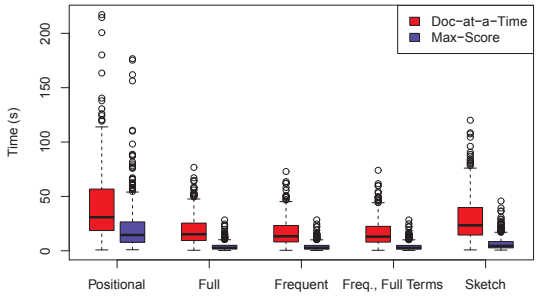
(a) GOV2, Long queries, SDM



(b) ClueWeb-09-B, Long queries, SDM



(c) GOV2, Long queries, WSDM-Int



(d) ClueWeb-09-B, Long queries, WSDM-Int

Figure 8.2: Average query execution times for long queries (4 to 12 terms). On each graph the query processing times are measured for both the Document-at-a-Time and Max-Score algorithms. Graphs shown span two collections (GOV2 and ClueWeb-09-B), two models (SDM, and WSDM-Int). Each query processing time is the average of 5 repeated executions.

12 terms. This separation is based on a query log analysis (Bendersky and Croft, 2009). Importantly, Bendersky and Croft (2009) show that longer queries can have very different properties than shorter queries. We execute each set of these queries on the 5 sets of index structures, using both Document-at-a-Time and Max-Score algorithms.

Figure 8.1 shows the distribution of query processing times for the short query set, and Figure 8.2 shows the distribution of query processing times for the long query set. In each figure, results are displayed for each collection, and each algorithm.

We can see a number of trends in this data. First, we can see that the frequent, frequent-with-full-terms and sketch indexes all improve query retrieval performance considerably over positional indexes. Both frequent indexes are comparable to the full index for query execution speed. The sketch index exhibits slightly slower query execution speeds.

Aggregating these results over both query sizes, both collections, and both query processing algorithms, we observe there is no significant difference in retrieval efficiency between the frequent index and the frequent, with full terms index.

We observe that the frequent index is able to execute queries approximately 66% faster than the positional index, and 6% faster than the full index. The sketch index is able to execute queries 44% faster than the positional index, and 57% slower than the full index.

We also observe that Document-at-a-Time consistently requires more processing time than Max-Score. Indeed, across all scenarios of collections, retrieval models, query sizes, and index data structures, in these experiments, we observe a reduction in processing time of 60%.

As observed in Chapter 4, WSDM-Int is a considerably more complex model than SDM, as it requires the aggregation of a range of statistics to determine the appropriate weight for each term and window feature. In these experiments, we

Table 8.1: Space requirements of different sets of index structures. All indexes are able to execute SDM and WSDM-Int. Freq., Full terms is a hybrid index that retains all terms, but discards infrequent ordered and unordered windows.

Index	Vocab. (GB)	Postings (GB)	Combined (GB)
GOV2			
Positional	0.41	42.9	43.3
Full	33.5	261	294
Freq.,	3.07	231	234
Freq., with Full term	3.45	232	235
Sketch	0.98	466	467
ClueWeb-09-Cat-B			
Positional	0.41	59.5	59.9
Full	47.9	431	479
Freq.,	5.88	389	395
Freq., with Full term	6.24	390	396
Sketch	1.08	774	775

observe a very small difference in the efficiency of each of these retrieval models. SDM can be processed just 3.5% faster than WSDM-Int, as averaged over all scenarios.

8.3.2 Space Requirements

Table 8.1 shows a comparison of the space requirements for each index, for the GOV2 and ClueWeb-09-B collections. We can see that the vocabulary data of the frequent indexes requires just 10% of the space required to store the vocabulary of the full index. The postings data stored in the frequent indexes, however, requires around 90% of the space required to store the postings data of the full index.

While the vocabulary space requirements of the sketch index is reduced to approximately 2% of the vocabulary space requirements of the full index, the posting list data requires around 78% more space than the full index. As observed in Chapter 7, the sketch index is appropriate where the space required to store the vocabulary is larger than the space required to store the postings data.

Research presented in Chapters 6 and 7 shows that the indexing of larger n-grams furthers improves the space savings possible, with respect to the full index.

8.4 Summary

In this chapter, we have empirically investigated the efficiency of the proposed term dependency indexes for the evaluation of two strong dependency retrieval models. The efficiency of the term dependency index structures were compared over two different query execution models, for both long and short queries, as executed on two large collections.

We observed that for the SDM and WSDM-Int retrieval models, both the frequent index and the sketch index considerably improve retrieval efficiency. The frequent index is able to reduce query processing time, relative to a positional index, by an average of 66%. Further, the frequent index improves retrieval efficiency over the full index by 6%. The sketch index also exhibits considerable performance improvements, reducing query processing time by 44%, relative to the positional index.

The total space requirements of the frequent index, for the required term dependency, is 78% of the space requirements of the full index. The total space requirements for the sketch index, however, are larger than the full index for these retrieval models. This observation was also made in Chapter 7, where the sketch index was observed to be a space-efficient index structure for term dependencies that contain more than just 2 terms.

As discussed in Chapter 2, several improvements to the Max-Score algorithm have been proposed. First, Strohman et al. (2005) presents an optimization of this algorithm that uses sets of `topdocs` to improve the contribution bounds for each retrieval model feature. Each set of `topdocs` is the set of documents with the highest scores for the associated feature. A second improvement to the Max-Score algorithm is presented by Turtle et al. (1996). This algorithm uses sampling methods to more accurately compute the maximum and minimum score bounds that are used to determine which concepts must be scored to determine if a document should be included in the current top k documents, or not, at the time of processing. In future

research should determine relative benefits of each of these MAX-SCORE improvements with respect to the term dependency indexes investigated in this thesis.

An alternate query processing algorithm, not evaluated in this chapter, is the WEAK-AND algorithm (Broder et al., 2003). This query processing algorithm allows a threshold controlled interpolation between boolean `OR` and `AND` query processing. The WEAK-AND algorithm operates by selecting which documents to fully score using a combination of the upper and lower score bounds and a threshold score determined by the k^th highest document score. To the best of our knowledge there is no publication directly comparing WEAK-AND to MAX-SCORE, an interesting direction for future work would be to determine under which circumstances one algorithm is preferred over the other.

Finally, both the frequent and the sketch index structures provide parameter controlled trade-offs between space requirements, retrieval efficiency and retrieval effectiveness. In this chapter, we specifically selected parameters that ensure accurate retrieval effectiveness was achieved. This measure allowed the direct comparison of query processing times between these structures and the two baseline index structures. An investigation of the three-way trade-off between retrieval efficiency, space requirements and retrieval effectiveness for each of these structures, for high performing dependency models is important to conduct.

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

9.1 Conclusions

In this body of work, we investigated the problem of improving the retrieval efficiency of dependency retrieval models, while minimizing space requirements, and without compromising retrieval effectiveness. As part of this study we have: conducted a systematic comparison of a wide range of proximity-based dependency models; investigated the space and time efficiency of existing indexing solutions; proposed two new index structures for dependency features; and investigated benefits offered by these structures for the most effective dependency retrieval models.

In Chapter 4, we performed a systematic comparison of state-of-the-art bi-term dependency models. This comparison was subject to a number of restrictions. Specifically, we restricted the comparison to models that use proximity-based dependencies between sequentially extracted sets of queried terms, that do not require external data sources, and do not require the use of pseudo-relevance feedback algorithms. We also proposed new many-term dependency models, based on strong performing bi-term dependency models. We performed a systematic comparison of many-term dependency models, using the strongest performing bi-term models as benchmark models. Additionally, we provided tuned parameters for a wide range of popular dependency models, for three standard test collections.

Our results support previous findings that bi-term dependency models can consistently outperform bag-of-words models. We observe that dependency models produce the largest improvements over bag-of-words models on longer queries. The best per-

forming bi-term model, given the restrictions applied, is a variant of the weighted sequential dependence model. Our experiments also show that many-term dependency models do not consistently outperform bi-term models. However, per-query analysis shows that many-term proximity features have some potential to improve retrieval performance, if used in a more selective manner.

In Chapter 5, we discussed existing index structures for term dependency data. We detailed modifications to existing index construction algorithms that enable the efficient and scalable construction of each type of index. The positional and full index structures form a baseline against which the frequent index and sketch index data structures are compared.

We note that there are several possible algorithms that combine positional posting lists, stored in the positional index. We investigated the application of three assumptions that could allow for a trade-off between retrieval efficiency and effectiveness, in the extraction of window instances from positional posting lists. Empirically, we find that there is no reason to prefer any particular type of extraction algorithm. While the extracted collection statistics varied between the algorithms, the retrieval efficiency, and retrieval effectiveness did not noticeably change across the set of algorithms.

We also investigated the distributions of a range of different types of windows that may be indexed using a full index. We measured the skew in vocabulary and the growth rates for a range of window parameters. This data allowed us to make predictions about the vocabulary sizes, and space requirements of full indexes, for a range of window types. We verified the accuracy of these predictions by comparison to actual full indexes.

In Chapter 6, we proposed and investigated a new index structure, the frequent index. We explored the problem of constructing an index of frequent n-grams for large English corpora. Our new Location-Based Window Two-Pass algorithm provided a new useful blend of attributes, in that it required less than half the amount

of temporary disk space of the Disk-Based Frequent Window One-Pass approach, while requiring less than twice as much processing time. This represents a significant benefit in terms of practical usefulness. We demonstrated that this approach can readily be adapted for use across a cluster of computers, and is scalable in this distributed sense, a virtue that more than compensates for its slower execution speed.

We empirically evaluated the two best performing algorithms for the indexing of frequent unordered windows, of width 8, containing 2 terms. We observed that both the Location-Based Window Two-Pass algorithm, and the Disk-Based Frequent Window One-Pass algorithm are both scalable in both an monolithic and a parallel sense. However, the space saving benefits of the Location-Based Window Two-Pass algorithm were observed to be minimal for this type of window.

We investigated the relationship between the threshold and the space required by the index structure. We observed that the largest space reductions, relative to a full index, occurs at the lowest threshold values $h < 5$. The majority of these observed space savings resulted from a large reduction in the size of the vocabulary, making this technique considerably more effective for many-term dependencies.

We also analyzed how the frequent index structure affects retrieval effectiveness through two experiments. First, we analyzed two query logs to determine the fraction of queries that would be affected by a range of threshold settings. We observed that only a small fraction of queries in the query log are affected by small threshold settings ($h < 100$). We measured retrieval effectiveness using annotated TREC collections. This experiment demonstrated that small threshold values ($h < 100$), generally ensure that retrieval effectiveness is unchanged relative to a full index of term dependencies.

In Chapter 7, we investigated the problem of accurately estimating document and collection level term dependency statistics in large data collections. The sketch index was presented as a novel approach to estimating n-gram statistics for infor-

mation retrieval tasks. By using frequency sketching techniques developed for data streaming applications, we were able to accurately estimate collection and document level statistics, and provide an attractive trade-off between space and relative error. Furthermore, we showed how to bound the space usage of the data structure.

We demonstrated that our approach is efficient in both time and space requirements, and can provide a low error rate for all of the statistics being estimated. Empirically, we showed that space requirements of the vocabulary component of a sketch index of 1-to-4-grams, extracted from the GOV2 collection, is reduced to less than 0.01% of the requirements of an equivalent full index. We observed that sketch indexes can process queries considerably faster than both positional indexes, and next-word indexes. Unlike frequent indexes and query-log cache approaches, our method does not require an auxiliary index to calculate statistics for unseen or infrequent n-grams. This new index representation provides an attractive alternative to other state-of-the-art approaches depending on many-term dependency statistics for retrieval tasks.

Finally, in Chapter 8, we empirically investigated the efficiency of the proposed term dependency indexes for the evaluation of two strong dependency retrieval models. The efficiency of the term dependency index structures were compared over two different query execution models, for both long and short queries, as executed on two large collections.

We observed that for the SDM and WSDM-Int retrieval models, both the frequent index and the sketch index considerably improve retrieval efficiency. The frequent index was able to reduce query processing time, relative to a positional index, by an average of 66%. Further, the frequent index improved retrieval efficiency over the full index by 6%. The sketch index also exhibited considerable performance improvements, reducing query processing time by 44%, relative to the positional index.

The total space requirements of the frequent index, for the required term dependency, was just 78% of the space requirements of the full index. The total space requirements for the sketch index, however, were larger than the full index for these retrieval models. This observation was also made in Chapter 7, where the sketch index was observed to be a space-efficient index structure for term dependencies that contain more than just 2 terms.

In conclusion, in this thesis, we have performed an extensive comparison of dependency retrieval models. We have investigated existing data structures designed to store and retrieve statistics for the strongest dependency models. We have proposed and analyzed two new index structures, the frequent index and the sketch index, designed to store and retrieve dependency statistics. Additionally, we have empirically tested the application of these structures to the strongest performing dependency retrieval models and observed that the novel index structures proposed in this thesis offer attractive new trade-offs between space requirements and retrieval efficiency.

9.2 Future Work

In Chapter 4, we performed an extensive comparison of proximity-based dependency retrieval models. There are two important extensions to this work, to investigate in future work. First, an important problem for information retrieval is determining the portability of each of these models. In enterprise or personal search, systems must be deployed and retrieve information or documents from unseen collections. Importantly, a sufficient quantity of training data may not be available to enable appropriate tuning of model parameters. A study into the portability of retrieval models should consider both cases where there is no training data available, and where there is an insufficient quantity of training data.

A second extension to the work presented in Chapter 4 is an investigation into the utility of external data sources for dependency retrieval models. This extension is the

relaxation of some of the constraints placed upon selection of dependency retrieval models in that chapter. Previous work has shown that external data sources have the potential to greatly improve information retrieval (Bendersky et al., 2012). However, it is vital to compare these benefits to the strongest performing models that do not use external data sources.

In Chapter 5, we observed stopword-like window instances in the top 10 most frequent term dependencies (see Table 5.1). In this thesis, we did not omit any of these stopword-like windows from any indexes. Omitting these stopword-like windows could dramatically reduce the space requirements of posting list data, for full, frequent and sketch index structures. However, analysis should include an evaluation of the impact that the omission of these windows would have on retrieval performance. A variety of strategies are available to determine the set of stop-windows, for example, measuring the fraction of stopwords in the window.

In Chapter 6, we investigated construction algorithms for frequent indexes of term dependencies. We argued that the Hash-Based Window Two-Pass algorithm is not scalable in a parallel sense. This is because it requires that a hash table large enough to store a frequency for each window in the collection be stored in memory on each computing node. To ensure minimal collisions, the hash table must grow with the size of the collection, so, this approach is not scalable in parallel. An alternative that we didn't consider in Chapter 6, is to use a **CountMin** sketch in place of the hash table. In Chapter 7, we observed that this structure grows sub-linearly with the size of the collection, while supporting a specific error rate. So, use of this hash-based structure can reduce the memory requirements on each node to grow sub-linearly with the collection. Importantly, this would permit distributed indexing of very large collections, even with limited per-processor memory space.

In Chapter 7, we proposed and investigated the sketch index, as an approximate index of term dependencies. We observed that this structure considerably reduces

the space requirements of vocabulary data. However, the space required by stored postings data can increase. Multi-dimensional sketching techniques, such as those presented by Thaper et al. (2002), may be an effective method of sketching posting list data. A problem for this technique is that the posting lists are used in query processing algorithms to determine which documents contain at least one of the query terms. Sketching techniques would eliminate this efficiency improvement. Another alternative is to use a secondary CountMin sketch to approximate the data stored in each posting list. This option requires that all data in each required posting list is read into memory for the execution of a query. If this approach is used, we can consider performing matrix-like operations to simultaneously combine document statistics for each query term and term dependency, and thereby, score each document in the collection. However, a final iteration over the documents is still required to collect and return the top k documents.

Finally, in Chapter 8 we analyzed the application of each of these index structures for term dependencies to the efficient execution of two strong dependency models, SDM and WSDM-Internal. We did not investigate the efficiency of alternative query execution algorithms, such as Weak-And (Broder et al., 2003), for dependency models. To the best of our knowledge, the efficiency of the Max-Score algorithm has not been directly compared to the Weak-And algorithm. Future research should determine which algorithm is preferable, in which circumstances.

APPENDIX

DETAILS OF PROXIMITY-BASED MODEL COMPARISON

In Chapter 4, we presented retrieval model performance across three collections, and two types of queries. Retrieval model performance was measured as the average performance across 5 folds, for each collection and query set. This appendix details each of the retrieval models, the retrieval model parameters learned for each query fold, the performance of each query fold, and the p-values from all statistical comparisons. The data presented in this appendix enables the reproducibility of all reported results in Chapter 4.

Query Folds

The details for each fold is shown here in a series of Tables:

Collection	Query Set	Table
Robust-04	Titles	Table A.1
Robust-04	descriptions	Table A.2
GOV2	Titles	Table A.3
GOV2	descriptions	Table A.4
ClueWeb-09-Cat-B	Titles	Table A.5
ClueWeb-09-Cat-B	descriptions	Table A.6

Table A.1: Query folds for Robust-04 topic Titles, each cell is a TREC topic id.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
1	302	301	306	320	304
2	303	308	307	325	305
3	309	312	313	330	310
4	316	322	321	332	311
5	317	327	324	335	314
6	319	328	326	337	315
7	323	338	334	342	318
8	331	343	347	344	329
9	336	348	351	350	333
10	341	349	354	355	339
11	356	352	358	368	340
12	357	360	361	377	345
13	370	364	362	379	346
14	373	365	363	387	353
15	378	369	376	393	359
16	381	371	380	398	366
17	383	374	382	402	367
18	392	386	396	405	372
19	394	390	404	407	375
20	406	397	413	408	384
21	410	403	415	412	385
22	411	419	417	420	388
23	414	422	427	421	389
24	426	423	436	425	391
25	428	424	437	430	395
26	433	432	439	431	399
27	447	434	444	435	400
28	448	440	445	438	401
29	601	446	449	616	409
30	607	602	450	618	416
31	608	604	603	625	418
32	612	611	605	630	429
33	617	623	606	633	441
34	619	624	614	636	442
35	635	627	620	639	443
36	641	632	622	649	609
37	642	638	626	650	610
38	646	643	628	653	613
39	647	651	631	655	615
40	654	652	637	657	621
41	656	663	644	659	629
42	662	674	648	667	634
43	665	675	661	668	640
44	669	678	664	672	645
45	670	680	666	673	658
46	679	683	671	676	660
47	684	688	677	682	681

Continued on next page

Table A.1 – *Continued from previous page*

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
48	690	689	685	686	694
49	692	695	687	691	696
50	700	698	693	697	699

Table A.2: Query folds for Robust-04 topic descriptions, each cell is a TREC topic id.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
1	309	301	302	308	307
2	312	303	305	313	311
3	315	304	306	314	319
4	317	324	310	325	322
5	320	329	316	327	334
6	326	335	318	331	336
7	328	337	321	341	348
8	330	343	323	344	352
9	332	349	333	345	356
10	340	353	338	350	357
11	342	359	339	351	362
12	346	361	358	355	363
13	347	373	371	360	374
14	354	379	375	364	376
15	365	380	383	367	384
16	366	381	387	368	390
17	370	382	388	369	394
18	372	385	392	386	408
19	377	389	414	395	410
20	378	396	415	397	412
21	391	401	419	398	422
22	393	407	420	402	425
23	399	416	431	404	430
24	400	428	435	409	440
25	403	429	437	411	442
26	405	433	439	413	443
27	406	436	441	417	444
28	423	446	601	418	602
29	426	449	603	421	619
30	427	607	605	424	620
31	432	608	606	450	622
32	434	612	611	609	624
33	438	614	617	613	633
34	445	616	627	621	636
35	447	623	635	626	641
36	448	625	637	631	643
37	604	628	644	632	653
38	610	629	648	639	655
39	615	630	652	640	656
40	618	646	660	642	657

Continued on next page

Table A.2 – *Continued from previous page*

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
41	634	649	664	647	658
42	638	651	665	661	659
43	645	654	666	662	670
44	650	663	669	667	672
45	671	679	675	668	673
46	674	682	676	683	678
47	677	691	688	687	680
48	681	695	690	689	684
49	686	696	692	693	685
50	697	698	694	700	699

Table A.3: Query folds for GOV2 topic Titles, each cell is a TREC topic id.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
1	712	704	702	701	711
2	722	708	703	713	716
3	731	709	705	714	719
4	740	715	706	717	739
5	749	718	707	723	741
6	750	738	710	726	742
7	751	743	720	727	744
8	759	755	721	729	745
9	764	756	724	732	746
10	765	761	725	735	748
11	774	762	728	760	752
12	775	763	730	769	753
13	782	772	733	778	754
14	784	779	734	780	758
15	785	792	736	787	766
16	786	802	737	790	767
17	788	806	747	791	768
18	789	808	757	793	776
19	794	809	770	795	777
20	798	814	771	799	781
21	801	816	773	811	783
22	805	818	797	813	796
23	823	821	803	819	800
24	824	822	804	826	807
25	825	831	810	827	815
26	832	836	812	837	817
27	833	839	820	842	829
28	835	840	828	846	830
29	845	843	834	848	838
30	850	847	844	849	841

Table A.4: Query folds for GOV2 topic descriptions, each cell is a TREC topic id.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
1	701	710	707	705	702
2	703	716	715	706	708
3	704	720	718	711	709
4	713	730	723	712	714
5	733	735	727	717	721
6	734	742	728	719	725
7	738	745	729	722	731
8	739	752	740	724	743
9	744	758	741	726	747
10	750	759	746	732	748
11	753	761	751	736	749
12	757	764	779	737	754
13	763	776	782	755	760
14	768	777	784	756	762
15	770	781	786	765	766
16	772	791	788	771	767
17	773	801	789	787	769
18	775	806	794	790	774
19	778	810	795	796	792
20	780	812	803	797	798
21	783	821	805	802	799
22	785	822	813	807	800
23	793	829	819	809	811
24	804	837	823	820	814
25	808	839	827	824	816
26	815	841	830	828	826
27	817	842	832	835	831
28	818	843	834	840	836
29	825	844	845	846	838
30	833	847	849	848	850

Table A.5: Query folds for ClueWeb-09-Cat-B title topics, each cell is a TREC query id.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
1	1	2	3	5	4
2	6	7	15	10	13
3	8	9	17	11	16
4	25	14	18	12	24
5	27	20	19	28	29
6	35	21	22	31	39
7	36	23	30	42	48
8	41	26	32	50	52
9	53	33	38	59	64
10	54	34	43	63	65
11	55	37	46	66	72

Continued on next page

Table A.5 – *Continued from previous page*

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
12	57	40	47	69	90
13	58	44	49	74	91
14	60	45	61	78	96
15	62	51	70	79	97
16	73	56	76	84	98
17	75	67	80	95	104
18	92	68	81	101	106
19	93	71	82	108	107
20	94	77	85	118	112
21	100	83	86	122	114
22	102	89	87	124	119
23	105	99	88	129	121
24	117	110	103	132	123
25	120	111	109	135	126
26	125	115	113	145	127
27	128	116	131	147	134
28	130	138	136	148	140
29	133	142	137	149	143
30	141	144	139	150	146

Table A.6: Query folds for GOV2 description topics, each cell is a TREC query id.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
1	12	2	9	3	1
2	14	4	15	13	6
3	16	5	29	24	7
4	18	10	36	33	8
5	19	11	39	35	17
6	26	21	43	40	20
7	27	22	48	46	25
8	28	23	55	50	38
9	30	34	59	52	41
10	31	37	68	56	42
11	32	53	69	58	44
12	47	60	70	63	45
13	51	61	73	75	49
14	54	62	88	79	74
15	57	67	90	81	80
16	64	72	91	86	83
17	65	76	96	92	89
18	66	78	100	101	93
19	71	82	102	109	95
20	77	84	103	110	97
21	87	85	106	111	99
22	94	98	108	116	105
23	107	104	112	121	113
24	118	124	114	130	115
25	122	138	119	131	117

Continued on next page

Table A.6 – *Continued from previous page*

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
26	125	140	120	133	123
27	132	141	135	139	126
28	134	144	136	142	127
29	137	145	146	143	128
30	147	150	149	148	129

Learned Parameter Settings

In this section, we provide a mathematical definition of each of the retrieval models that was investigated in Chapter 4. We also detail the learned settings for the parameters for each retrieval model, for each fold, collection, and query set. We present retrieval models in the following order:

1. query likelihood (QL);
2. the sequential dependence model (SDM);
3. SDM variant: Uni+O234;
4. SDM variant: Uni+O234+U2;
5. SDM variant: Uni+O23+U23;
6. SDM variant: Uni+O234+U234;
7. the weighted sequential dependence model (WSDM);
8. WSDM variant: WSDM-Int;
9. WSDM variant: WSDM-Int-3;
10. PLM;
11. PLM-2;
12. PL2;
13. pDFR-BiL2;
14. pDFR-PL2;
15. BM25;
16. BM25-TP;
17. BM25-TP2; and
18. BM25-Span.

Query Likelihood

The ranking function for the Dirichlet smoothed, Query Likelihood model (Song and Croft, 1999) is defined as:

$$P(D_j|Q) \stackrel{rank}{=} \sum_{q_i \in Q} \frac{tf_{q_i,D_j} + \mu \cdot \frac{cf_{q_i}}{|C|}}{|D_j| + \mu}$$

where μ is the smoothing parameter. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Rob-04	Titles	μ	935	852	844	878	1161	934	885
Rob-04	Desc.	μ	2103	2397	2108	2102	2118	2166	2107
GOV2	Titles	μ	1446	1462	1525	1474	1496	1481	1477
GOV2	Desc.	μ	2087	1973	2288	2091	2094	2107	2121
Clue-B	Titles	μ	2300	2141	3130	2637	2528	2547	2531
Clue-B	Desc.	μ	1925	2128	2026	2070	2011	2032	2084

Sequential Dependence Model

The ranking function for the sequential dependence model (SDM) (Metzler and Croft, 2005) is defined as:

$$\begin{aligned}
P(D|Q) &\stackrel{\text{rank}}{=} \sum_{c \in C(Q)} \lambda_C f_C(c) \\
&= \sum_{c \in T} \lambda_T f_T(c) + \sum_{c \in O} \lambda_O f_O(c) + \sum_{c \in U} \lambda_U f_U(c) \\
T &= \{q_i \in Q\} \\
O &= U = \{(q_i, q_{i+1}) \in Q\} \\
f_T(c) &= \log P(q_i|D) \\
f_O(c) &= \log P(\#od1(q_i, q_{i+1})|D) \\
f_U(c) &= \log P(\#uw8(q_i, q_{i+1})|D) \\
P(x|D) &= \frac{tf_{x,D} + \mu \frac{tf_x}{|C|}}{|D| + \mu}
\end{aligned}$$

where μ is the Dirichlet smoothing parameter, and the combination of features is controlled by three parameters, λ_T, λ_O and λ_U. $\#od1$ is the ordered window operator that matches adjacent terms in documents, $\#uw8$ is a unordered window operator that matches pairs of terms that occur within a window of 8 terms in documents. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Rob-04	Titles	μ	957	977	1893	999	1520	1270	1136
Rob-04	Titles	λ_T	0.884	0.883	0.856	0.869	0.871	0.873	0.86
Rob-04	Titles	λ_O	0.0674	0.0852	0.0844	0.089	0.0693	0.0791	0.0786
Rob-04	Titles	λ_U	0.049	0.0321	0.0591	0.0418	0.0594	0.0483	0.0612
Rob-04	Desc.	μ	2711	3575	2749	3465	2801	3060	3050
Rob-04	Desc.	λ_T	0.813	0.82	0.838	0.816	0.815	0.82	0.829
Rob-04	Desc.	λ_O	0.134	0.102	0.086	0.103	0.129	0.111	0.105
Rob-04	Desc.	λ_U	0.0533	0.0782	0.0762	0.0811	0.056	0.069	0.0657
GOV2	Titles	μ	2052	2076	1870	1876	1865	1948	1937
GOV2	Titles	λ_T	0.869	0.853	0.86	0.859	0.873	0.863	0.861
GOV2	Titles	λ_O	0.0433	0.0644	0.0498	0.0507	0.0502	0.0517	0.0513
GOV2	Titles	λ_U	0.0876	0.0824	0.09	0.0903	0.0766	0.0854	0.0881
GOV2	Desc.	μ	3283	3214	2966	3003	3601	3213	3070
GOV2	Desc.	λ_T	0.856	0.858	0.841	0.839	0.841	0.847	0.837
GOV2	Desc.	λ_O	0.0961	0.0791	0.101	0.102	0.114	0.0986	0.102
GOV2	Desc.	λ_U	0.0481	0.0631	0.0573	0.0588	0.0447	0.0544	0.0604
Clue-B	Titles	μ	3840	2698	6391	4328	4300	4311	4318
Clue-B	Titles	λ_T	0.875	0.883	0.737	0.859	0.874	0.846	0.859
Clue-B	Titles	λ_O	0.0575	0.0461	0.051	0.0561	0.0546	0.0531	0.056
Clue-B	Titles	λ_U	0.0672	0.0704	0.212	0.0852	0.0709	0.101	0.0852
Clue-B	Desc.	μ	2408	2683	3455	2580	2345	2694	2578
Clue-B	Desc.	λ_T	0.864	0.902	0.873	0.886	0.884	0.882	0.883
Clue-B	Desc.	λ_O	0.0359	0.00846	0.0355	0.0172	0.0156	0.0225	0.0117
Clue-B	Desc.	λ_U	0.0997	0.0893	0.0911	0.0964	0.101	0.0954	0.105

Sequential Dependence Model Variant: Uni+O234

The ranking function for the Uni+O234 variant of the sequential dependence model (Metzler and Croft, 2005) is defined as:

$$P(D|Q) \stackrel{\text{rank}}{=} \sum_{c \in C(Q)} \lambda_C f_C(c)$$

$$= \sum_{c \in T} \lambda_T f_T(c) + \sum_{c \in O} \lambda_{O2} f_{O2}(c) + \sum_{c \in O3} \lambda_{O3} f_{O3}(c) + \sum_{c \in O4} \lambda_{O4} f_{O4}(c)$$

$$T = \{q_i \in Q\}$$

$$O2 = \{(q_i, q_{i+1}) \in Q\}$$

$$O3 = \{(q_i, q_{i+1}, q_{i+2}) \in Q\}$$

$$O4 = \{(q_i, q_{i+1}, q_{i+2}, q_{i+3}) \in Q\}$$

$$f_T(c) = \log P(q_i|D)$$

$$f_{O2}(c) = \log P(\#od1(q_i, q_{i+1})|D)$$

$$f_{O3}(c) = \log P(\#od1(q_i, q_{i+1}, q_{i+2})|D)$$

$$f_{O4}(c) = \log P(\#od1(q_i, q_{i+1}, q_{i+2}, q_{i+3})|D)$$

$$P(x|D) = \frac{tf_{x,D} + \mu \frac{tf_x}{|C|}}{|D| + \mu}$$

where μ is the smoothing parameter, and the combination of features is controlled by four parameters, λ_T, λ_{O2}, λ_{O3} and λ_{O4}. $\#od1$ is the ordered window operator that matches adjacent terms in documents. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Robust-04								
Titles	μ	918	885	1292	1325	1500	1184	1074
Titles	λ_T	1.029	0.869	0.894	0.827	0.971	0.918	0.911
Titles	λ_{O2}	0.107	0.097	0.121	0.113	0.129	0.113	0.116
Titles	λ_{O3}	0.045	0.039	-0.011	0.05	0.078	0.04	0.014
Titles	λ_{O4}	-0.18	-0.0057	-0.0027	0.0099	-0.18	-0.072	-0.042
Desc.	μ	2999	3252	2790	3503	3627	3234	3624
Desc.	λ_T	0.786	0.734	0.837	0.817	0.8	0.795	0.774
Desc.	λ_{O2}	0.131	0.125	0.143	0.13	0.129	0.132	0.13
Desc.	λ_{O3}	0.083	0.059	0.0	0.047	0.07	0.052	0.083
Desc.	λ_{O4}	0.0	0.083	0.02	0.005	0.0017	0.022	0.013
GOV2								
Titles	μ	1816	2215	1952	1571	1789	1868	1799
Titles	λ_T	0.93	0.877	0.903	0.899	0.932	0.908	0.908
Titles	λ_{O2}	0.071	0.088	0.084	0.055	0.072	0.074	0.068
Titles	λ_{O3}	0.0	0.014	0.003	0.0093	-0.01	0.0031	0.0
Titles	λ_{O4}	-0.00095	0.021	0.0098	0.036	0.0058	0.014	0.023
Desc.	μ	3234	2837	3793	3337	3277	3296	3286

Continued on next page

Continued from previous page

Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Desc.	λ_T	0.88	0.893	0.847	0.873	0.878	0.874	0.877
Desc.	λ_{O2}	0.104	0.108	0.098	0.1	0.117	0.106	0.105
Desc.	λ_{O3}	0.016	0.014	0.045	0.029	0.014	0.024	0.023
Desc.	λ_{O4}	0.0	-0.016	0.01	-0.0018	-0.0099	-0.0035	-0.0051
ClueWeb-09-Cat-B								
Titles	μ	2896	3363	5142	3597	3123	3624	3624
Titles	λ_T	0.936	0.885	0.908	0.918	0.918	0.913	0.912
Titles	λ_{O2}	0.059	0.051	0.069	0.06	0.058	0.059	0.059
Titles	λ_{O3}	0.023	0.05	0.023	0.033	0.014	0.028	0.028
Titles	λ_{O4}	-0.017	0.015	0.0	-0.011	0.0099	-0.00064	0.0
Desc.	μ	1888	1900	2474	1868	1927	2012	1883
Desc.	λ_T	0.927	0.925	0.923	0.932	0.942	0.93	0.95
Desc.	λ_{O2}	0.064	0.039	0.066	0.064	0.036	0.054	0.045
Desc.	λ_{O3}	0.0027	0.026	0.018	0.0032	0.022	0.014	0.022
Desc.	λ_{O4}	0.0063	0.0099	-0.0065	0.0	0.0	0.0019	-0.017

Sequential Dependence Model Variant: Uni+O234+U2

The ranking function for the Uni+O234+U2 variant of the sequential dependence model (Metzler and Croft, 2005) is defined as:

$$
\begin{aligned}
P(D|Q) &\stackrel{\text{rank}}{=} \sum_{c \in C(Q)} \lambda_C f_C(c) \\
&= \sum_{c \in T} \lambda_T f_T(c) + \sum_{c \in O} \lambda_{O2} f_{O2}(c) + \sum_{c \in O3} \lambda_{O3} f_{O3}(c) \\
&\quad + \sum_{c \in O4} \lambda_{O4} f_{O4}(c) + \sum_{c \in U2} \lambda_{U2} f_{U2}(c) \\
T &= \{q_i \in Q\} \\
O2 = U2 &= \{(q_i, q_{i+1}) \in Q\} \\
O3 &= \{(q_i, q_{i+1}, q_{i+2}) \in Q\} \\
O4 &= \{(q_i, q_{i+1}, q_{i+2}, q_{i+3}) \in Q\} \\
f_T(c) &= \log P(q_i|D) \\
f_{O2}(c) &= \log P(\#od1(q_i, q_{i+1})|D) \\
f_{O3}(c) &= \log P(\#od1(q_i, q_{i+1}, q_{i+2})|D) \\
f_{O4}(c) &= \log P(\#od1(q_i, q_{i+1}, q_{i+2}, q_{i+3})|D) \\
f_{U2}(c) &= \log P(\#uw8(q_i, q_{i+1})|D) \\
P(x|D) &= \frac{tf_{x,D} + \mu \frac{tf_x}{|C|}}{|D| + \mu}
\end{aligned}
$$

where μ is the smoothing parameter, and the combination of features is controlled by five parameters, λ_T, λ_{O2}, λ_{O3}, λ_{O4}, and λ_{U2}. $\#od1$ is the ordered window operator that matches adjacent terms in documents, and $\#uw8$ is a unordered window operator that matches pairs of terms that occur within a window of 8 terms in documents. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Robust-04								
Titles	μ	1381	1366	1580	1070	1643	1408	1326
Titles	λ_T	0.846	0.815	0.842	0.814	0.808	0.825	0.823
Titles	λ_{O2}	0.046	0.06	0.067	0.088	0.061	0.065	0.06
Titles	λ_{O3}	0.016	0.053	0.0049	0.057	0.071	0.04	0.061
Titles	λ_{O4}	0.02	0.018	0.029	-0.0043	0.0099	0.014	0.0027
Titles	λ_{U2}	0.071	0.053	0.057	0.045	0.05	0.055	0.054
Desc.	μ	2957	4060	2894	2996	3704	3322	3466
Desc.	λ_T	0.782	0.722	0.772	0.811	0.731	0.763	0.727
Desc.	λ_{O2}	0.1	0.081	0.088	0.109	0.093	0.094	0.092
Desc.	λ_{O3}	0.128	0.0	0.054	0.035	0.12	0.067	0.05

Continued on next page

Continued from previous page

Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Desc.	λ_{O4}	-0.065	0.145	0.018	0.0	0.0036	0.02	0.075
Desc.	λ_{U2}	0.056	0.052	0.068	0.046	0.052	0.055	0.056
GOV2								
Titles	μ	1920	2452	1889	1833	1922	2003	2013
Titles	λ_T	0.859	0.838	0.857	0.848	0.875	0.856	0.857
Titles	λ_{O2}	0.049	0.068	0.055	0.046	0.055	0.055	0.054
Titles	λ_{O3}	0.0097	0.004	0.014	0.011	-0.007	0.0064	0.0063
Titles	λ_{O4}	-0.005	0.0092	-0.0098	0.0097	0.0015	0.0011	0.0011
Titles	λ_{U2}	0.088	0.081	0.083	0.085	0.075	0.082	0.081
Desc.	μ	3671	3293	3442	3386	3933	3545	3466
Desc.	λ_T	0.832	0.848	0.826	0.846	0.825	0.835	0.836
Desc.	λ_{O2}	0.084	0.073	0.09	0.082	0.085	0.083	0.087
Desc.	λ_{O3}	0.018	0.033	0.018	0.029	0.037	0.027	0.017
Desc.	λ_{O4}	0.018	-0.019	0.013	-0.023	0.0016	-0.0019	0.0053
Desc.	λ_{U2}	0.048	0.065	0.053	0.066	0.051	0.057	0.054
ClueWeb-09-Cat-B								
Titles	μ	5502	3803	5350	6267	4785	5141	3662
Titles	λ_T	0.844	0.853	0.73	0.851	0.866	0.829	0.889
Titles	λ_{O2}	0.059	0.038	0.046	0.043	0.036	0.044	0.033
Titles	λ_{O3}	0.029	0.049	0.024	0.031	0.03	0.032	0.025
Titles	λ_{O4}	-0.0061	0.0098	-0.003	0.0	0.0018	0.0005	0.0
Titles	λ_{U2}	0.075	0.05	0.204	0.076	0.066	0.094	0.054
Desc.	μ	2349	2756	2966	2033	2736	2568	2636
Desc.	λ_T	0.891	0.872	0.862	0.871	0.866	0.872	0.859
Desc.	λ_{O2}	0.009	0.024	0.036	0.056	0.032	0.031	0.037
Desc.	λ_{O3}	0.02	-0.01	-0.014	-0.014	-0.01	-0.0056	-0.012
Desc.	λ_{O4}	0.0049	0.012	0.016	0.0039	0.0076	0.0087	0.019
Desc.	λ_{U2}	0.076	0.102	0.101	0.084	0.104	0.093	0.097

Sequential Dependence Model Variant: Uni+O23+U23

The ranking function for the Uni+O23+U23 variant of the sequential dependence model (Metzler and Croft, 2005) is defined as:

$$
\begin{aligned}
P(D|Q) &\stackrel{\text{rank}}{=} \sum_{c \in C(Q)} \lambda_C f_C(c) \\
&= \sum_{c \in T} \lambda_T f_T(c) + \sum_{c \in O} \lambda_{O2} f_{O2}(c) + \sum_{c \in O3} \lambda_{O3} f_{O3}(c) \\
&\quad + \sum_{c \in U2} \lambda_{U2} f_{U2}(c) + \sum_{c \in U3} \lambda_{U3} f_{U3}(c) \\
T &= \{q_i \in Q\} \\
O2 = U2 &= \{(q_i, q_{i+1}) \in Q\} \\
O3 = U3 &= \{(q_i, q_{i+1}, q_{i+2}) \in Q\} \\
f_T(c) &= \log P(q_i|D) \\
f_{O2}(c) &= \log P(\#od1(q_i, q_{i+1})|D) \\
f_{O3}(c) &= \log P(\#od1(q_i, q_{i+1}, q_{i+2})|D) \\
f_{U2}(c) &= \log P(\#uw8(q_i, q_{i+1})|D) \\
f_{U3}(c) &= \log P(\#uw12(q_i, q_{i+1}, q_{i+2})|D) \\
P(x|D) &= \frac{tf_{x,D} + \mu \frac{tf_x}{|C|}}{|D| + \mu}
\end{aligned}
$$

where μ is the smoothing parameter, and the combination of features is controlled by five parameters, λ_T, λ_{O2}, λ_{O3}, λ_{U2}, and λ_{U3}. $\#od1$ is the ordered window operator that matches adjacent terms in documents, $\#uw8$ is a unordered window operator that matches pairs of terms that occur within a window of 8 terms in documents, and $\#uw12$ is a unordered window operator that matches sets of three terms that occur within a window of 12 terms in documents. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Robust-04								
Titles	μ	1562	1500	2178	1800	1863	1781	1500
Titles	λ_T	0.827	0.855	0.825	0.803	0.85	0.832	0.863
Titles	λ_{O2}	0.045	0.077	0.09	0.057	0.095	0.073	0.059
Titles	λ_{O3}	0.029	0.0097	0.0069	0.021	0.011	0.016	0.02
Titles	λ_{U2}	0.054	0.048	0.052	0.049	0.031	0.047	0.059
Titles	λ_{U3}	0.045	0.0097	0.025	0.071	0.013	0.033	0.0
Desc.	μ	3591	4247	2907	4068	3215	3606	3606
Desc.	λ_T	0.736	0.734	0.754	0.689	0.753	0.733	0.719
Desc.	λ_{O2}	0.095	0.087	0.087	0.078	0.085	0.086	0.084
Desc.	λ_{O3}	0.085	0.078	0.042	0.13	0.104	0.088	0.096
Desc.	λ_{U2}	0.043	0.044	0.032	0.047	0.024	0.038	0.047

Continued on next page

Continued from previous page

Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Desc.	λ_{U3}	0.041	0.057	0.086	0.056	0.034	0.055	0.054
GOV2								
Titles	μ	2790	2170	2300	2300	2300	2372	2032
Titles	λ_T	0.867	0.853	0.88	0.835	0.856	0.858	0.831
Titles	λ_{O2}	0.059	0.064	0.048	0.043	0.05	0.053	0.041
Titles	λ_{O3}	0.0049	0.0	-0.016	0.0099	-0.015	-0.0034	-0.00082
Titles	λ_{U2}	0.069	0.074	0.055	0.073	0.063	0.067	0.074
Titles	λ_{U3}	0.0	0.0095	0.034	0.039	0.046	0.026	0.054
Desc.	μ	4060	3695	4103	3964	3460	3856	3682
Desc.	λ_T	0.865	0.776	0.782	0.848	0.845	0.823	0.807
Desc.	λ_{O2}	0.077	0.073	0.078	0.107	0.107	0.088	0.088
Desc.	λ_{O3}	0.019	0.029	0.038	-0.021	0.0	0.013	0.01
Desc.	λ_{U2}	0.029	0.048	0.057	0.033	0.038	0.041	0.045
Desc.	λ_{U3}	0.0095	0.074	0.044	0.033	0.0099	0.034	0.05
ClueWeb-09-Cat-B								
Titles	μ	6630	3560	7791	7254	4800	6007	6700
Titles	λ_T	0.839	0.824	0.709	0.856	0.812	0.808	0.847
Titles	λ_{O2}	0.047	0.037	0.028	0.045	0.055	0.042	0.049
Titles	λ_{O3}	0.019	0.034	0.016	0.008	0.0078	0.017	0.015
Titles	λ_{U2}	0.086	0.047	0.215	0.074	0.066	0.097	0.079
Titles	λ_{U3}	0.0095	0.06	0.033	0.018	0.059	0.036	0.01
Desc.	μ	2992	3896	2780	2930	3616	3243	4016
Desc.	λ_T	0.851	0.895	0.953	0.864	0.868	0.886	0.898
Desc.	λ_{O2}	0.04	0.012	0.044	0.047	0.091	0.047	0.011
Desc.	λ_{O3}	0.0	0.019	-0.015	-0.013	-0.057	-0.013	0.021
Desc.	λ_{U2}	0.072	0.031	0.017	0.039	0.04	0.04	0.034
Desc.	λ_{U3}	0.037	0.043	0.0	0.063	0.058	0.04	0.036

Sequential Dependence Model Variant: Uni+O234+U234

The ranking function for the Uni+O234+U234 variant of the sequential dependence model (Metzler and Croft, 2005) is defined as:

$$
\begin{aligned}
P(D|Q) &\stackrel{\text{rank}}{=} \sum_{c \in C(Q)} \lambda_C f_C(c) \\
&= \sum_{c \in T} \lambda_T f_T(c) + \sum_{c \in O} \lambda_{O2} f_{O2}(c) + \sum_{c \in O3} \lambda_{O3} f_{O3}(c) + \sum_{c \in O4} \lambda_{O4} f_{O4}(c) \\
&\quad + \sum_{c \in U2} \lambda_{U2} f_{U2}(c) + \sum_{c \in U3} \lambda_{U3} f_{U3}(c) + \sum_{c \in U4} \lambda_{U4} f_{U4}(c) \\
T &= \{q_i \in Q\} \\
O2 = U2 &= \{(q_i, q_{i+1}) \in Q\} \\
O3 = U3 &= \{(q_i, q_{i+1}, q_{i+2}) \in Q\} \\
O4 = U4 &= \{(q_i, q_{i+1}, q_{i+2}, q_{i+3}) \in Q\} \\
f_T(c) &= \log P(q_i|D) \\
f_{O2}(c) &= \log P(\#od1(q_i, q_{i+1})|D) \\
f_{O3}(c) &= \log P(\#od1(q_i, q_{i+1}, q_{i+2})|D) \\
f_{O4}(c) &= \log P(\#od1(q_i, q_{i+1}, q_{i+2}, q_{i+3})|D) \\
f_{U2}(c) &= \log P(\#uw8(q_i, q_{i+1})|D) \\
f_{U3}(c) &= \log P(\#uw12(q_i, q_{i+1}, q_{i+2})|D) \\
f_{U4}(c) &= \log P(\#uw16(q_i, q_{i+1}, q_{i+2}, q_{i+3})|D) \\
P(x|D) &= \frac{tf_{x,D} + \mu \frac{tf_x}{|C|}}{|D| + \mu}
\end{aligned}
$$

where μ is the smoothing parameter, and the combination of features is controlled by five parameters, λ_T, λ_{O2}, λ_{O3}, λ_{O4}, λ_{U2}, λ_{U3}, and λ_{U4}. $\#od1$ is the ordered window operator that matches adjacent terms in documents, $\#uw8$ is a unordered window operator that matches pairs of terms that occur within a window of 8 terms in documents, $\#uw12$ is a unordered window operator that matches sets of three terms that occur within a window of 12 terms in documents, and $\#uw16$ is a unordered window operator that matches sets of four terms that occur within a window of 16 terms in documents. We did not learn parameters for title queries for this model. This is because almost no title queries contain four or more terms, so the optimal parameter settings are nearly identical to Uni+O23+u23, detailed above. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Robust-04								

Continued on next page

Continued from previous page

Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Desc.	μ	4069	3501	3724	3643	3765	3740	2788
Desc.	λ_T	0.671	0.804	0.71	0.701	0.717	0.721	0.802
Desc.	λ_{O2}	0.105	0.093	0.087	0.079	0.082	0.089	0.109
Desc.	λ_{O3}	0.04	0.018	0.042	0.083	0.096	0.056	0.043
Desc.	λ_{O4}	0.118	0.018	0.0	0.0098	0.012	0.032	-0.0083
Desc.	λ_{U2}	0.034	0.053	0.052	0.051	0.032	0.044	0.013
Desc.	λ_{U3}	0.033	0.02	0.089	0.075	0.034	0.05	0.041
Desc.	λ_{U4}	-0.0012	-0.0058	0.019	0.00047	0.027	0.008	0.0
GOV2								
Desc.	μ	3674	2998	4060	4452	3809	3798	4508
Desc.	λ_T	0.851	0.859	0.878	0.817	0.825	0.846	0.826
Desc.	λ_{O2}	0.075	0.06	0.091	0.074	0.106	0.081	0.086
Desc.	λ_{O3}	0.022	-0.004	0.0	0.013	0.014	0.0091	0.0094
Desc.	λ_{O4}	-0.0019	-0.0052	0.0038	0.012	-0.016	-0.0015	0.00034
Desc.	λ_{U2}	0.033	0.032	0.028	0.043	0.024	0.032	0.036
Desc.	λ_{U3}	0.023	0.051	0.0	0.013	0.026	0.023	0.035
Desc.	λ_{U4}	-0.0031	0.0063	0.0	0.028	0.021	0.01	0.008
ClueWeb-09-Cat-B								
Desc.	μ	2860	2952	2495	2595	4560	3092	2880
Desc.	λ_T	0.91	0.798	0.912	0.932	0.815	0.873	0.933
Desc.	λ_{O2}	0.029	0.013	0.022	0.012	0.054	0.026	0.039
Desc.	λ_{O3}	0.0078	0.0054	-0.00063	-0.0071	-0.032	-0.0054	-0.012
Desc.	λ_{O4}	0.0	0.012	-0.0019	-0.0012	0.012	0.0042	0.0081
Desc.	λ_{U2}	0.065	0.097	0.072	0.05	0.067	0.07	0.037
Desc.	λ_{U3}	0.0	0.073	0.0057	0.014	0.067	0.032	0.0
Desc.	λ_{U4}	-0.012	0.0018	-0.0089	0.0012	0.017	-0.00019	-0.004

Weighted Sequential Dependence Model

The ranking function for the weighted sequential dependence model (Bendersky et al., 2010) is defined as:

$$
\begin{aligned}
P(Q|D) &\overset{\text{rank}}{=} \sum_{c \in C(Q)} \lambda_C f_C(c) \\
&= \sum_{c \in T} \lambda_T(c) f_T(c) + \sum_{c \in O} \lambda_O(c) f_O(c) + \sum_{c \in U} \lambda_U(c) f_U(c) \\
T &= \{q_i \in Q\} \\
O &= U = \{(q_i, q_{i+1}) \in Q\} \\
f_T(c) &= \log P(q_i|D) \\
f_O(c) &= \log P(\#od1(q_i, q_{i+1})|D) \\
f_U(c) &= \log P(\#uw8(q_i, q_{i+1})|D) \\
P(x|D) &= \frac{tf_{x,D} + \mu \frac{tf_x}{|C|}}{|D| + \mu} \\
\lambda_T(q_i) &= \sum_{j \in K_1} w_j^1 \cdot g_j^1(q_i) \\
\lambda_O(q_i, q_{i+1}) &= \lambda_U(q_i, q_{i+1}) = \sum_{j \in K_2} w_j^2 \cdot g_j^2(q_i, q_{i+1})
\end{aligned}
$$

where μ is the smoothing parameter. The weight of each term and bi-term feature is determined as the weighted sum of a set of term or bi-term features. The set of feature functions are defined as:

$$g_1^1(t) = c^1 = 1$$
$$g_2^1(t) = cf^1(t)$$
$$g_3^1(t) = dc^1(t)$$
$$g_4^1(t) = Gcf^1(t)$$
$$g_5^1(t) = Mcf^1(t)$$
$$g_5^1(t) = Wcf^1(t)$$
$$g_1^2(t_i, t_j) = c^2 = 1$$
$$g_2^2(t_i, t_j) = cf^2(\#od1(t_i, t_j))$$
$$g_3^2(t_i, t_j) = dc^2(\#od1(t_i, t_j))$$
$$g_4^2(t_i, t_j) = Gcf^2(\#od1(t_i, t_j))$$
$$g_5^2(t_i, t_j) = Mcf^2(\#od1(t_i, t_j))$$
$$g_5^2(t_i, t_j) = Wcf^2(\#od1(t_i, t_j))$$

where c is the constant 1; cf is a function that returns the collection frequency of the term or bigram in the target collection; dc is a function that returns the number of documents in which the term or bigram occurs, in the target collection; Gcf is a function that returns the frequency of the term or bigram in the Google-n-grams collection; Mcf is a function that returns the frequency of the term or bigram in the MSN query log; and Wcf is a function that returns the frequency of the term or bigram in the set of all Wikipedia titles. $\#od1$ is the ordered window operator that matches adjacent terms in documents, $\#uw8$ is a unordered window operator that matches pairs of terms that occur within a window of 8 terms in documents. There are 13 parameters for this model, the smoothing parameter μ, 6 weights (w_j^1) for each of the term statistic functions (g_j^1), and 6 weights (w_j^2) for each of the bigram statistic functions (g_j^2). The learned settings for these parameters for each collection and query set fold are listed in the following table.

Param.	1	2	3	4	5	Avg.	Oracle
Robust-04, Topic Titles							
μ	1500	1500	1780	1500	1490	1554	1500
w_c^1	0.863	0.863	0.713	0.863	0.863	0.833	0.863
w_{dc}^1	-0.085	-0.085	-0.076	-0.085	-0.085	-0.083	-0.085
w_{cf}^1	0.03	0.03	0.03	0.03	0.029	0.03	0.03
w_{Gcf}^1	0.0	0.0	0.002	0.0	0.0	0.0004	0.0
w_{Mcf}^1	0.0	0.0	0.001	0.0	0.001	0.0004	0.0
w_{Wcf}^1	0.0	0.0	0.0	0.0	0.002	0.0004	0.0
w_c^2	0.049	0.049	0.049	0.049	0.049	0.049	0.049
w_{dc}^2	-0.001	-0.001	-0.001	-0.001	-0.001	-0.001	-0.001
w_{cf}^2	0.0042	0.0042	0.0043	0.0042	0.0042	0.0042	0.0042
w_{Gcf}^2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
w_{Mcf}^2	0.0	0.0	0.001	0.004	0.004	0.0018	0.0
w_{Wcf}^2	0.0	0.0	0.0	-0.001	-0.001	-0.0004	0.0
Robust-04, Topic Descriptions							
μ	1580	1790	1800	1830	1850	1770	1820
w_c^1	0.823	0.703	0.693	0.693	0.623	0.707	0.683
w_{dc}^1	-0.091	-0.078	-0.079	-0.075	-0.077	-0.08	-0.076
w_{cf}^1	0.025	0.031	0.03	0.031	0.031	0.029	0.029
w_{Gcf}^1	-0.002	0.0	-0.001	0.0	0.001	-0.0004	-0.001
w_{Mcf}^1	0.005	0.001	0.006	0.0	0.0	0.0024	0.0
w_{Wcf}^1	0.01	0.003	0.001	-0.001	0.01	0.0046	0.006
w_c^2	0.039	0.049	0.049	0.049	0.049	0.047	0.049
w_{dc}^2	0.0	-0.003	-0.001	-0.001	-0.002	-0.0014	-0.001
w_{cf}^2	0.0032	0.0043	0.0043	0.0043	0.0023	0.0037	0.0043
w_{Gcf}^2	0.0	0.001	-0.001	0.0	0.0	0.0	0.0
w_{Mcf}^2	0.004	0.001	0.004	0.004	0.004	0.0034	0.004
w_{Wcf}^2	0.005	0.0	0.004	0.0	0.0	0.0018	-0.001
GOV2, Topic Titles							
μ	2069	2109	2089	1818	2109	2039	2099
w_c^1	0.783	0.793	0.793	0.787	0.793	0.789	0.783
w_{dc}^1	-0.05	-0.051	-0.05	-0.026	-0.05	-0.046	-0.052
w_{cf}^1	0.022	0.024	0.023	0.00022	0.023	0.019	0.023
w_{Gcf}^1	-0.002	-0.001	0.0	-2.6e-05	0.0	-0.00061	0.0
w_{Mcf}^1	0.001	0.001	0.0	0.0015	0.0	0.0007	0.002
w_{Wcf}^1	0.007	-0.001	0.0	0.00091	0.0	0.0014	0.0
w_c^2	0.049	0.049	0.049	0.082	0.049	0.056	0.049
w_{dc}^2	0.0029	0.0019	0.0019	-7.3e-06	0.0019	0.0017	0.0019
w_{cf}^2	-0.0026	-0.0016	-0.0016	-0.0046	-0.0016	-0.0024	-0.0016
w_{Gcf}^2	0.0	0.0	0.0	-8.1e-05	0.0	-1.6e-05	0.0
w_{Mcf}^2	-0.001	0.003	0.001	0.0026	0.0	0.0011	0.001
w_{Wcf}^2	0.0	-0.002	0.001	0.00038	0.0	-0.00012	0.0
GOV2, Topic Descriptions							
μ	2780	2910	2760	3080	3361	2978	2489
w_c^1	0.57	0.76	0.71	0.523	0.88	0.688	0.763
w_{dc}^1	-0.037	-0.018	-0.035	-0.059	-0.044	-0.039	-0.054
w_{cf}^1	-0.002	-0.014	-0.007	0.027	-0.00081	0.00059	0.01
w_{Gcf}^1	-0.004	-0.017	-0.004	-0.003	-0.014	-0.0084	-0.001
w_{Mcf}^1	0.012	0.001	0.009	-0.001	0.009	0.006	0.002

Continued on next page

Continued from previous page

Param.	1	2	3	4	5	Avg.	Oracle
w^1_{Wcf}	0.01	0.025	0.003	0.014	0.012	0.013	0.006
w^2_c	0.01	0.03	0.02	0.049	0.04	0.03	0.039
w^2_{dc}	0.0	-0.001	0.0	-0.002	-0.0025	-0.0011	-0.00014
w^2_{cf}	-0.001	-0.004	-0.002	-0.0014	6.6e-05	-0.0017	-0.0036
w^2_{Gcf}	0.001	0.004	0.001	0.0	0.001	0.0014	0.001
w^2_{Mcf}	-0.001	-0.003	0.001	-0.002	0.003	-0.0004	0.0
w^2_{Wcf}	0.005	0.004	0.003	0.006	0.004	0.0044	0.005
ClueWeb-09-Cat-B, Topic Titles							
μ	3461	3471	6260	4118	4204	4303	4050
w^1_c	0.88	0.88	0.79	0.795	0.644	0.798	0.79
w^1_{dc}	-0.029	-0.031	0.001	0.0011	-9.4e-05	-0.011	-0.005
w^1_{cf}	0.00042	-0.00075	0.002	-0.00062	0.00018	0.00024	0.0
w^1_{Gcf}	-0.001	0.0	-0.032	0.00015	-0.00077	-0.0067	0.005
w^1_{Mcf}	-0.001	0.0	0.0	-0.00087	-5.4e-05	-0.00038	-0.034
w^1_{Wcf}	0.0	0.0	-0.002	0.0012	-1.8e-05	-0.00017	-0.001
w^2_c	0.08	0.07	0.02	0.088	0.074	0.067	0.01
w^2_{dc}	-0.00097	-0.00097	0.001	-2.4e-05	-0.00053	-0.0003	0.002
w^2_{cf}	0.0006	0.00037	0.0	0.00046	0.00042	0.00037	0.008
w^2_{Gcf}	0.0	0.0	0.001	0.0013	0.0003	0.00051	-0.002
w^2_{Mcf}	0.0	0.0	0.0	0.00027	-0.00015	2.4e-05	-0.004
w^2_{Wcf}	-0.007	0.0	-0.004	-0.00062	0.0007	-0.0022	0.001
ClueWeb-09-Cat-B, Topic Descriptions							
μ	3401	3134	1940	4150	4050	3335	3431
w^1_c	0.87	0.804	0.703	0.81	0.863	0.81	0.84
w^1_{dc}	-0.049	-0.054	-0.084	-0.032	-0.089	-0.062	-0.047
w^1_{cf}	-0.00057	-0.002	0.029	-0.01	0.046	0.012	-0.0017
w^1_{Gcf}	-0.001	0.00034	-0.004	0.0	-0.008	-0.0025	0.0
w^1_{Mcf}	0.002	0.013	0.033	-0.002	0.001	0.0093	0.001
w^1_{Wcf}	0.0	-5e-05	0.0	-0.002	0.0	-0.00041	0.0
w^2_c	0.06	0.082	0.039	0.01	0.029	0.044	0.04
w^2_{dc}	-0.0043	-0.0063	-0.003	0.0	-0.002	-0.0031	-0.0021
w^2_{cf}	-0.00011	-0.00049	0.0043	0.0	0.0003	0.0008	-0.001
w^2_{Gcf}	0.0	0.00028	-0.003	0.0	0.0	-0.00054	0.0
w^2_{Mcf}	0.002	0.0058	0.004	0.004	0.004	0.004	0.004
w^2_{Wcf}	0.005	9.5e-05	-0.001	0.002	0.0	0.0012	0.0

Weighted Sequential Dependence Model, Internal Variant

The ranking function for the internal variant of the weighted sequential dependence model (Bendersky et al., 2010) is defined as:

$$
\begin{aligned}
P(Q|D) &\overset{\text{rank}}{=} \sum_{c \in C(Q)} \lambda_C f_C(c) \\
&= \sum_{c \in T} \lambda_T(c) f_T(c) + \sum_{c \in O} \lambda_O(c) f_O(c) + \sum_{c \in U} \lambda_U(c) f_U(c) \\
T &= \{q_i \in Q\} \\
O &= U = \{(q_i, q_{i+1}) \in Q\} \\
f_T(c) &= \log P(q_i|D) \\
f_O(c) &= \log P(\#od1(q_i, q_{i+1})|D) \\
f_U(c) &= \log P(\#uw8(q_i, q_{i+1})|D) \\
P(x|D) &= \frac{tf_{x,D} + \mu \frac{tf_x}{|C|}}{|D| + \mu} \\
\lambda_T(q_i) &= \sum_{j \in K_1} w_j^1 \cdot g_j^1(q_i) \\
\lambda_O(q_i, q_{i+1}) &= \lambda_U(q_i, q_{i+1}) = \sum_{j \in K_2} w_j^2 \cdot g_j^2(q_i, q_{i+1})
\end{aligned}
$$

where μ is the smoothing parameter. The weight of each term and bi-term feature is determined as the weighted sum of a set of term or bi-term features. The set of feature functions are defined as:

$$
\begin{aligned}
g_1^1(t) &= c^1 = 1 \\
g_2^1(t) &= cf^1(t) \\
g_3^1(t) &= dc^1(t) \\
g_1^2(t_i, t_j) &= c^2 = 1 \\
g_2^2(t_i, t_j) &= cf^2(\#od1(t_i, t_j)) \\
g_3^2(t_i, t_j) &= dc^2(\#od1(t_i, t_j))
\end{aligned}
$$

where c is the constant 1; cf is a function that returns the collection frequency of the term or bigram in the target collection; and dc is a function that returns the number of documents in which the term or bigram occurs, in the target collection. $\#od1$ is the ordered window operator that matches adjacent terms in documents, $\#uw8$ is a unordered window operator that matches pairs of terms that occur within a window of 8 terms in documents.There are 7 parameters for this model, the smoothing parameter μ, 3 weights (w_j^1) for each of the term statistic functions (g_j^1), and 3 weights (w_j^2) for each of the bigram statistic functions (g_j^2). The learned settings for these parameters for each collection and query set fold are listed in the following table.

Param.	1	2	3	4	5	Avg.	Oracle
Robust-04, Topic Titles							
μ	1530	1477	1487	2099	1467	1612	1500
w_c^1	0.873	0.651	0.651	0.793	0.571	0.707	0.863
w_{dc}^1	-0.085	-0.035	-0.033	-0.049	-0.033	-0.047	-0.085
w_{cf}^1	0.031	-0.0078	-0.0068	0.00031	-0.0068	0.0019	0.03
w_c^2	0.049	0.029	0.019	0.039	0.019	0.031	0.049
w_{dc}^2	0.004	-0.0015	0.0005	0.0039	-0.0015	0.0011	-0.001
w_{cf}^2	-0.0028	0.0061	0.0071	0.0013	0.0071	0.0038	0.0042
Robust-04, Topic Descriptions							
μ	2089	1520	1467	1490	1540	1621	1790
w_c^1	0.793	0.804	0.651	0.804	0.873	0.785	0.703
w_{dc}^1	-0.049	-0.048	-0.033	-0.054	-0.093	-0.055	-0.075
w_{cf}^1	0.00035	-0.0028	-0.0068	0.0012	0.03	0.0043	0.031
w_c^2	0.039	0.056	0.019	0.016	0.049	0.036	0.049
w_{dc}^2	0.0029	-0.0021	0.00065	0.0039	0.0	0.0011	-0.001
w_{cf}^2	0.0033	0.0053	0.0071	0.0043	0.0042	0.0048	0.0043
GOV2, Topic Titles							
μ	2180	2331	2130	1820	1830	2058	2109
w_c^1	0.81	0.827	0.81	0.853	0.913	0.842	0.793
w_{dc}^1	-0.018	-0.038	-0.017	-0.068	-0.069	-0.042	-0.05
w_{cf}^1	-0.008	0.006	-0.003	0.031	0.038	0.013	0.023
w_c^2	0.0	0.076	0.02	0.049	0.069	0.043	0.049
w_{dc}^2	0.009	0.00044	0.008	0.002	0.001	0.0041	0.0019
w_{cf}^2	-0.005	-0.0022	-0.005	-0.0038	-0.0028	-0.0038	-0.0016
GOV2, Topic Descriptions							
μ	3481	2810	2729	2770	3519	3062	2780
w_c^1	0.8	0.833	0.813	0.863	0.743	0.81	0.804
w_{dc}^1	-0.037	-0.074	-0.043	-0.079	-0.058	-0.058	-0.045
w_{cf}^1	-0.0017	0.031	0.00016	0.031	0.016	0.015	0.0032
w_c^2	0.07	0.039	0.039	0.059	0.029	0.048	0.026
w_{dc}^2	-0.0033	0.0	-0.0011	-0.002	-0.00026	-0.0013	-0.0021
w_{cf}^2	0.00034	0.00017	0.00033	-0.00031	0.0013	0.00036	0.0043
ClueWeb-09-Cat-B, Topic Titles							

Continued on next page

Continued from previous page

Param.	1	2	3	4	5	Avg.	Oracle
μ	3471	3451	3491	4317	4435	3833	3461
w_c^1	0.88	0.88	0.88	1.291	0.807	0.947	0.88
w_{dc}^1	-0.031	-0.03	-0.029	-0.032	2.8e-05	-0.024	-0.029
w_{cf}^1	-0.00061	-0.0017	-0.0017	-0.0058	0.0003	-0.0019	-0.00047
w_c^2	0.07	0.07	0.06	0.079	0.092	0.074	0.06
w_{dc}^2	-0.00097	-0.00097	-0.00097	0.0005	-0.00044	-0.00057	-0.00097
w_{cf}^2	0.0002	0.00029	0.0002	0.0011	0.0009	0.00054	0.00037
ClueWeb-09-Cat-B, Topic Descriptions							
μ	1650	1820	2149	3561	2780	2392	2877
w_c^1	0.814	0.824	0.783	0.68	0.794	0.779	0.661
w_{dc}^1	-0.044	-0.044	-0.023	-0.03	-0.044	-0.037	-0.033
w_{cf}^1	-0.0028	-0.0028	-0.024	-0.0057	-0.00084	-0.0072	-0.0058
w_c^2	0.016	0.016	0.039	0.05	0.026	0.029	0.019
w_{dc}^2	-0.0031	-0.0031	-0.0031	-0.0035	-0.0031	-0.0032	-0.0025
w_{cf}^2	0.0053	0.0053	0.0011	0.00044	0.0033	0.0031	0.0031

Weighted Sequential Dependence Model Variant: WSDM-Int-3

The ranking function for the WSDM-Int-3 variant of the weighted sequential dependence model (Bendersky et al., 2010) is defined as:

$$
\begin{aligned}
P(Q|D) &\overset{\text{rank}}{=} \sum_{c \in C(Q)} \lambda_C f_C(c) \\
&= \sum_{c \in T} \lambda_T(c) f_T(c) + \sum_{c \in O2} \lambda_{O2}(c) f_O 2(c) + \sum_{c \in O3} \lambda_{O3}(c) f_O 3(c) \\
&\quad + \sum_{c \in U2} \lambda_{U2}(c) f_U 2(c) + \sum_{c \in U3} \lambda_{U3}(c) f_U 3(c) \\
T &= \{q_i \in Q\} \\
O2 = U2 &= \{(q_i, q_{i+1}) \in Q\} \\
O3 = U3 &= \{(q_i, q_{i+1}, q_{i+2}) \in Q\} \\
f_T(c) &= \log P(q_i|D) \\
f_{O2}(c) &= \log P(\#od1(q_i, q_{i+1})|D) \\
f_{O3}(c) &= \log P(\#od1(q_i, q_{i+1}, q_{i+2})|D) \\
f_{U2}(c) &= \log P(\#uw8(q_i, q_{i+1})|D) \\
f_{U3}(c) &= \log P(\#uw12(q_i, q_{i+1}, q_{i+2})|D) \\
P(x|D) &= \frac{tf_{x,D} + \mu \frac{tf_x}{|C|}}{|D| + \mu} \\
\lambda_T(q_i) &= \sum_{j \in K_1} w_j^1 \cdot g_j^1(q_i) \\
\lambda_O 2(q_i, q_{i+1}) = \lambda_U 2(q_i, q_{i+1}) &= \sum_{j \in K_2} w_j^2 \cdot g_j^2(q_i, q_{i+1}) \\
\lambda_O 3(q_i, q_{i+1}, q_{i+2}) = \lambda_U 3(q_i, q_{i+1}, q_{i+2}) &= \sum_{j \in K_3} w_j^3 \cdot g_j^3(q_i, q_{i+1}, q_{i+2})
\end{aligned}
$$

where μ is the smoothing parameter. The weight of each term, 2-term, and 3-gram feature is determined as the weighted sum of a set of term or bi-term features. The set of feature functions are defined as:

$$g_1^1(t) = c^1 = 1$$

$$g_2^1(t) = cf^1(t)$$

$$g_3^1(t) = dc^1(t)$$

$$g_1^2(t_i, t_j) = c^2 = 1$$

$$g_2^2(t_i, t_j) = cf^2(\#od1(t_i, t_j))$$

$$g_3^2(t_i, t_j) = dc^2(\#od1(t_i, t_j))$$

$$g_1^3(t_i, t_j, t_k) = c^3 = 1$$

$$g_2^3(t_i, t_j, t_k) = cf^3(\#od1(t_i, t_j, t_k))$$

$$g_3^3(t_i, t_j, t_k) = dc^3(\#od1(t_i, t_j, t_k))$$

where c is the constant 1; cf is a function that returns the collection frequency of the term, bigram or trigram in the target collection; and dc is a function that returns the number of documents in which the term, bigram or trigram occurs, in the target collection. $\#od1$ is the ordered window operator that matches adjacent sets of terms in documents, $\#uw8$ is a unordered window operator that matches pairs of terms that occur within a window of 8 terms in documents, and $\#uw12$ is a unordered window operator that matches sets of three terms that occur within a window of 12 terms in documents. There are 10 parameters for this model, the smoothing parameter μ, 3 weights (w_j^1) for each of the term statistic functions (g_j^1), 3 weights (w_j^2) for each of the bigram statistic functions (g_j^2), and 3 weights (w_j^3) for each of the trigram statistic functions (g_j^3). The learned settings for these parameters for each collection and query set fold are listed in the following table.

Param.	1	2	3	4	5	Avg.	Oracle
Robust-04, Topic Titles							
μ	1760	1510	1530	1510	1510	1564	1510
w_c^1	0.683	0.863	0.863	0.863	0.863	0.827	0.863
w_{dc}^1	-0.075	-0.085	-0.085	-0.085	-0.085	-0.083	-0.085

Continued on next page

Continued from previous page

Param.	1	2	3	4	5	Avg.	Oracle
w_{cf}^1	0.031	0.03	0.031	0.03	0.03	0.03	0.03
w_c^2	0.039	0.049	0.049	0.049	0.049	0.047	0.049
w_{dc}^2	-0.003	-0.001	-0.001	-0.001	-0.001	-0.0014	-0.001
w_{cf}^2	0.0043	0.0052	0.0062	0.0052	0.0052	0.0052	0.0052
w_c^3	-0.01	-0.01	0.0	-0.01	-0.01	-0.008	-0.01
w_{dc}^3	0.001	0.0	0.0	0.001	0.001	0.0006	0.001
w_{cf}^3	0.031	0.032	0.001	0.018	0.018	0.02	0.018
Robust-04, Topic Descriptions							
μ	1500	1790	1810	1800	1810	1742	1800
w_c^1	0.883	0.703	0.703	0.713	0.703	0.741	0.703
w_{dc}^1	-0.093	-0.075	-0.075	-0.075	-0.075	-0.079	-0.076
w_{cf}^1	0.031	0.03	0.031	0.03	0.031	0.03	0.031
w_c^2	0.049	0.049	0.049	0.039	0.039	0.045	0.039
w_{dc}^2	-0.001	-0.003	-0.001	0.0	-0.001	-0.0012	0.001
w_{cf}^2	0.0062	0.0053	0.0043	0.0053	0.0053	0.0053	0.0043
w_c^3	0.0	0.01	0.0	0.01	0.01	0.006	0.01
w_{dc}^3	0.008	0.007	0.0	0.0	0.009	0.0048	0.0
w_{cf}^3	-0.001	-0.001	-0.002	0.001	0.001	-0.0004	0.0
GOV2, Topic Titles							
μ	2099	2460	2191	2221	2150	2224	2130
w_c^1	0.823	0.663	0.87	0.89	0.923	0.833	0.643
w_{dc}^1	-0.051	-0.073	-0.03	-0.025	-0.068	-0.049	-0.088
w_{cf}^1	0.02	0.044	-0.00033	-0.005	0.032	0.018	0.064
w_c^2	0.069	0.061	0.05	0.07	0.059	0.062	0.059
w_{dc}^2	-6.6e-05	-0.005	-0.0021	-0.0033	-0.001	-0.0023	-0.002
w_{cf}^2	-0.0019	0.0033	0.0039	0.00044	0.00026	0.0012	-0.0004
w_c^3	0.02	0.01	0.03	0.05	0.02	0.026	0.02
w_{dc}^3	0.0	0.001	-0.001	-0.005	-0.001	-0.0012	-0.001
w_{cf}^3	-0.001	-0.002	0.0	0.0	-0.002	-0.001	0.0
GOV2, Topic Descriptions							
μ	3110	3030	2780	3471	3410	3160	3080
w_c^1	0.623	0.743	0.863	0.85	0.784	0.772	0.703
w_{dc}^1	-0.044	-0.059	-0.093	-0.037	-0.045	-0.056	-0.069
w_{cf}^1	0.015	0.023	0.044	-0.007	0.0022	0.015	0.031
w_c^2	0.039	0.029	0.029	0.04	0.026	0.033	0.039
w_{dc}^2	-0.004	-0.003	-0.004	-0.0013	-0.0021	-0.0029	-0.008
w_{cf}^2	0.0023	0.0043	0.0042	0.00082	0.0033	0.003	0.0063
w_c^3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
w_{dc}^3	0.0	0.0	-0.003	0.0	0.002	-0.0002	0.002
w_{cf}^3	0.002	0.0	0.004	0.002	-0.001	0.0014	-0.001
ClueWeb-09-Cat-B, Topic Titles							
μ	3461	2867	5796	6868	4050	4609	3140
w_c^1	0.87	0.691	0.485	0.812	0.16	0.603	1.323
w_{dc}^1	-0.029	-0.025	0.001	-0.00012	0.001	-0.01	-0.075
w_{cf}^1	-0.001	-0.0068	-0.0013	0.00027	-0.006	-0.003	0.03
w_c^2	0.06	0.029	0.091	0.083	0.01	0.055	0.079
w_{dc}^2	-0.0015	-0.0035	-0.0013	0.00064	0.0	-0.0011	-0.008
w_{cf}^2	0.00037	0.0021	-0.0051	0.0011	0.0	-0.00031	0.0053
w_c^3	0.02	0.01	0.038	0.067	0.0	0.027	-0.02

Continued on next page

Continued from previous page

Param.	1	2	3	4	5	Avg.	Oracle
w_{dc}^3	0.0	0.001	0.00093	-8.8e-05	0.0	0.00037	0.003
w_{cf}^3	0.002	0.001	-9.8e-05	0.00064	0.001	0.00091	0.008
ClueWeb-09-Cat-B, Topic Descriptions							
μ	3170	4070	3027	4370	2790	3485	2937
w_c^1	0.763	0.783	0.621	0.723	0.794	0.736	0.681
w_{dc}^1	-0.069	-0.085	-0.033	-0.084	-0.044	-0.063	-0.031
w_{cf}^1	0.031	0.045	-0.0058	0.047	0.0032	0.024	-0.0048
w_c^2	0.039	0.029	0.012	0.039	0.026	0.029	0.012
w_{dc}^2	0.0	-0.003	-0.0025	-0.005	-0.0061	-0.0033	-0.0035
w_{cf}^2	-0.0037	-3.6e-05	0.0021	0.0023	0.0043	0.00098	0.0031
w_c^3	0.01	0.0	0.0	0.01	0.01	0.006	0.0
w_{dc}^3	0.001	0.0	0.0	0.002	0.0	0.0006	0.001
w_{cf}^3	0.0	0.005	0.0	-0.004	0.001	0.0004	0.0

PLM

The ranking function for best-position PLM (Lv and Zhai, 2009) is defined as:

$$P(D|Q) = \max_{j<|D|} \left(\sum_{q_i \in Q} \frac{c'(q_i, j) + \mu \cdot \frac{cf_{q_i}}{|C|}}{Z_j + \mu} \right)$$

$$c'(q_i, j) = \sum_{k=1}^{|D|} c(w, k) \cdot k(k, j)$$

$$Z_j = \sum_{k=1}^{|D|} k(j, k)$$

$$k(j, k) = \exp\left(\frac{-(j-k)^2}{2\sigma^2} \right)$$

This formulation is the best-position strategy, using the Gaussian kernel. The parameters for the model are μ and σ. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Robust-04								
Titles	μ	632	517	748	894	891	737	741
Titles	σ	387.183	294.084	2664	1389	389.238	1025	566
Desc.	μ	3389	2734	3090	3190	3060	3093	3093
Desc.	σ	173.065	129.727	170.0	166.179	150.0	157.794	157.794
GOV2								
Titles	μ	632	517	748	894	891	737	741

Continued on next page

Continued from previous page

Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Titles	σ	387.183	294.084	2664	1389	389.238	1025	566
Desc.	μ	3389	2734	3090	3190	3060	3093	3093
Desc.	σ	173.065	129.727	170.0	166.179	150.0	157.794	157.794
ClueWeb-09-Cat-B								
Titles	μ	632	517	748	894	891	737	741
Titles	σ	387.183	294.084	2664	1389	389.238	1025	566
Desc.	μ	3389	2734	3090	3190	3060	3093	3093
Desc.	σ	173.065	129.727	170.0	166.179	150.0	157.794	157.794

PLM-2

The ranking function for PLM-2 (Lv and Zhai, 2009) is defined as:

$$P(D|Q) = \lambda \cdot \max_{j<|D|} \left(\sum_{q_i \in Q} \frac{c'(q_i, j) + \mu \cdot \frac{cf_{q_i}}{|C|}}{Z_j + \mu} \right) + (1 - \lambda) \cdot \left(\frac{tf_{q_i,D} + \mu \cdot \frac{cf_{q_i}}{|C|}}{|D| + \mu} \right)$$

$$c'(q_i, j) = \sum_{k=1}^{|D|} c(w, k) \cdot k(k, j)$$

$$Z_j = \sum_{k=1}^{|D|} k(j, k)$$

$$k(j, k) = \exp\left(\frac{-(j-k)^2}{2\sigma^2} \right)$$

This is the multi-σ strategy, using the Gaussian kernel. The second kernel is specified with σ_2 set to ∞, which is rank equivalent to the query likelihood model. The parameters for the model are μ and σ. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Robust-04								
Titles	λ	0.39	0.389	0.363	0.4	0.425	0.394	0.358
Titles	μ	934	1323	1199	690	1040	1037	873
Titles	σ	17.426	31.95	10.184	-30.0	-7.9	4.332	9.296
Desc.	λ	0.48	0.564	0.531	0.489	0.473	0.507	0.507
Desc.	μ	1649	1936	2083	2029	2250	1989	1989

Continued on next page

Continued from previous page

Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Desc.	σ	12.58	11.808	13.352	-11.0	12.328	7.782	11.674
GOV2								
Titles	λ	0.39	0.389	0.363	0.4	0.425	0.394	0.358
Titles	μ	934	1323	1199	690	1040	1037	873
Titles	σ	17.426	31.95	10.184	-30.0	-7.9	4.332	9.296
Desc.	λ	0.48	0.564	0.531	0.489	0.473	0.507	0.507
Desc.	μ	1649	1936	2083	2029	2250	1989	1989
Desc.	σ	12.58	11.808	13.352	-11.0	12.328	7.782	11.674
ClueWeb-09-Cat-B								
Titles	λ	0.39	0.389	0.363	0.4	0.425	0.394	0.358
Titles	μ	934	1323	1199	690	1040	1037	873
Titles	σ	17.426	31.95	10.184	-30.0	-7.9	4.332	9.296
Desc.	λ	0.48	0.564	0.531	0.489	0.473	0.507	0.507
Desc.	μ	1649	1936	2083	2029	2250	1989	1989
Desc.	σ	12.58	11.808	13.352	-11.0	12.328	7.782	11.674

PL2

The ranking function for the PL2 model (Amati and Van Rijsbergen, 2002) is defined as:

$$
\begin{aligned}
score_{PL2}(Q,D) = \sum_{q_i \in Q} \frac{1}{tfn_{q_i}+1} \cdot & \\
& \left(tfn_{q_i} \cdot \log_2\left(\frac{1}{\lambda_{q_i}}\right) + \frac{\lambda_{q_i}}{\log_2(e)} + \frac{1}{2} \cdot \log_2(2 \cdot \pi \cdot tfn_{q_i}) \right. \\
& \left. + tfn_{q_i} \cdot \left(\log_2(tfn_{q_i}) - \frac{1}{\log_2(e)} \right) \right) \\
\lambda_{q_i} = & \frac{cf_{q_i}}{|\mathcal{C}_\mathcal{D}|} \\
tfn_{q_i} = & tf_{q_i,D} \cdot log_2\left(1.0 + c \cdot \frac{|D_{avg}|}{|D|}\right)
\end{aligned}
$$

where $|\mathcal{C}_\mathcal{D}|$ is the number of documents in the collection and $|D_{avg}|$ is the average document length for the collection. This formulation is extracted from the canonical implementation of the model (Ounis et al., 2006). The only parameter of the model is c. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Rob-04	Titles	c	9.19	9.14	9.06	9.32	9.05	9.15	9.2
Rob-04	Desc.	c	2.14	2.17	2.1	2.09	2.38	2.18	2.09

Continued on next page

Continued from previous page

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
GOV2	Titles	c	7.94	6.76	7.95	7.22	7.93	7.56	7.96
GOV2	Desc.	c	3.85	4.31	3.83	4.3	3.92	4.04	3.94
Clue-B	Titles	c	19.1	15.4	15.6	15.4	18.3	16.8	16.9
Clue-B	Desc.	c	5.46	11.8	5.86	5.89	5.47	6.89	5.48

pDFR-BiL2

The ranking function for the pDFR-BiL2 model (Peng et al., 2007) is defined as:

$$score_{pDFR-BiL2}(D,Q) = \lambda \cdot score_{PL2}(D,Q) + (1-\lambda) \cdot score_{BiL2}(D,Q)$$

$$score_{BiL2}(D,Q) = \sum_{p \in Q} \frac{1}{pfn} \Bigg(- \log_2(|D|-1)! + \log_2(pfn)! - pfn \cdot \log_2 \left(\frac{1}{|D|-1} \right) - (|D|-1-pfn) \cdot \log_2 \left(1 - \frac{1}{|D|-1} \right) \Bigg)$$

$$pfn = tf_{p,D} \cdot \left(1 + c_p \cdot \frac{|D_{avg}|-1}{|D|-1} \right)$$

where p is a pair of adjacent terms extracted from the query, $tf_{p,D}$ is the frequency of the pair of terms in a document, as matched by an unordered window of width 5, and $|D_{avg}|$ is the average document length for the collection. This formulation of $score_{PL2}$ is specified above. There are three parameters for the model: c, c_p, and λ. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Rob-04	Titles	c	8.4	9.0	8.9	15.2	9.55	10.2	7.94
Rob-04	Titles	c_p	1649	216.0	828	54.1	209.0	591	225.0
Rob-04	Titles	λ	1.5	1.5	1.4	1.87	1.45	1.54	1.6
Rob-04	Desc.	c	1.66	1.44	1.81	2.18	2.2	1.86	1.46
Rob-04	Desc.	c_p	827	1667	222.0	436.0	6567	1944	1945
Rob-04	Desc.	λ	1.42	1.42	1.44	1.89	1.3	1.5	1.4
GOV2	Titles	c	6.91	7.81	6.67	6.66	7.79	7.17	7.17
GOV2	Titles	c_p	182.0	130.0	420.0	208.0	176.0	223.0	223.0
GOV2	Titles	λ	1.34	1.41	1.36	1.33	1.37	1.36	1.36
GOV2	Desc.	c	3.23	3.29	2.81	3.38	2.66	3.07	2.87
GOV2	Desc.	c_p	208.0	1648	591	1649	1944	1208	1208
GOV2	Desc.	λ	1.3	1.3	1.24	1.2	1.3	1.27	1.27
Clue-B	Titles	c	9.91	15.4	15.3	16.0	19.8	15.3	9.96
Clue-B	Titles	c_p	590	56.9	1947	7.38	1.0	520	1946

Continued on next page

Continued from previous page

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Clue-B	Titles	λ	1.14	1.25	1.2	0.974	1.0	1.11	1.1
Clue-B	Desc.	c	4.76	12.1	4.46	7.77	4.81	6.78	5.11
Clue-B	Desc.	c_p	1943	29.0	1944	29.5	591	907	14.5
Clue-B	Desc.	λ	1.1	0.993	1.1	0.954	1.04	1.04	0.987

pDFR-PL2

The ranking function for the pDFR-PL2 model, adapted from the pDFR-BiL2 model (Peng et al., 2007), is defined as:

$$
\begin{aligned}
score_{pDFR-BiL2}(D,Q) &= \lambda \cdot score_{PL2}(D,Q) + (1-\lambda) \cdot score_{pPL2}(D,Q) \\
score_{pPL2}(Q,D) &= \sum_{p \in Q} \frac{1}{pfn+1} \cdot \\
&\quad \left(pfn \cdot \log_2\left(\frac{1}{\lambda_p}\right) + \frac{\lambda_p}{\log_2(e)} + \frac{1}{2} \cdot \log_2(2 \cdot \pi \cdot pfn) \right. \\
&\quad \left. + pfn \cdot \left(\log_2(pfn) - \frac{1}{\log_2(e)} \right) \right) \\
\lambda_p &= \frac{cf_p}{|\mathcal{C}_\mathcal{D}|} \\
pfn &= tf_{p,D} \cdot log_2\left(1.0 + c \cdot \frac{|D_{avg}|}{|D|}\right)
\end{aligned}
$$

where p is a pair of adjacent terms extracted from the query; $tf_{p,D}$ is the frequency of the pair of terms in a document, as matched by an unordered window of width 5, cf_p is the collection frequency of the pair of terms p, $|C_D|$ is the number of documents in the collection, and $|D_{avg}|$ is the average document length for the collection. This formulation of $score_{PL2}$ is provided above. There are three parameters for the model: c, c_p, and λ. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Rob-04	Titles	c	8.89	9.25	9.8	7.85	9.05	8.97	9.9
Rob-04	Titles	c_p	30.9	4.2	13.8	24.3	13.4	17.3	14.6
Rob-04	Titles	λ	0.915	0.9	0.9	0.939	0.9	0.911	0.9
Rob-04	Desc.	c	2.4	2.35	2.15	2.78	2.75	2.49	2.4
Rob-04	Desc.	c_p	8.8	13.6	11.7	19.4	103.0	31.4	22.1
Rob-04	Desc.	λ	0.9	0.9	0.868	0.849	0.9	0.883	0.9
GOV2	Titles	c	11.9	11.4	15.0	11.1	10.4	12.0	12.0
GOV2	Titles	c_p	13.9	12.4	31.3	11.3	26.3	19.0	19.0
GOV2	Titles	λ	0.899	0.874	0.845	0.844	0.9	0.872	0.872

Continued on next page

Continued from previous page

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
GOV2	Desc.	c	4.3	4.24	3.84	4.84	4.25	4.3	4.4
GOV2	Desc.	c_p	10.3	52.5	14.0	14.0	17.8	21.7	21.8
GOV2	Desc.	λ	0.9	0.9	0.9	0.9	0.902	0.9	0.9
Clue-B	Titles	c	60.3	20.9	109.0	60.2	60.2	62.1	60.1
Clue-B	Titles	c_p	3.18	0.9	3.29	1.1	1.0	1.89	1.1
Clue-B	Titles	λ	0.889	0.9	0.84	0.9	0.9	0.886	0.9
Clue-B	Desc.	c	5.59	12.3	18.8	13.3	16.8	13.4	12.5
Clue-B	Desc.	c_p	30.7	1.2	0.3	0.371	1.4	6.79	6.59
Clue-B	Desc.	λ	0.883	0.9	0.8	0.898	0.8	0.856	0.856

BM25

The ranking function for the BM25 model (Robertson and Walker, 1994) is defined as:

$$RSV_{BM25}(Q,D) \stackrel{\text{rank}}{=} \sum_{q_i \in Q} \log \left[\frac{N}{dc_{q_i}} \right] \cdot \frac{(k_1+1) \cdot tf_{q_i,D}}{k_1 \left((1-b) + b \cdot \frac{|D|}{|D_{avg}|} \right) + tf_{t,D}}$$

where $|D_{avg}|$ is the average length of a document in the collection, and dc_{q_i} is the number of documents that contain the query term, q_i. The parameters for the model are b and k_1. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Rob-04	Titles	b	0.337	0.306	0.298	0.38	0.304	0.325	0.303
Rob-04	Titles	k_1	0.664	0.693	0.77	0.675	0.737	0.708	0.725
Rob-04	Desc.	b	0.532	0.591	0.675	0.49	0.509	0.559	0.67
Rob-04	Desc.	k_1	1.25	1.03	0.781	0.981	0.936	0.996	0.892
GOV2	Titles	b	0.317	0.344	0.326	0.391	0.313	0.338	0.335
GOV2	Titles	k_1	0.864	0.929	0.873	0.91	0.952	0.906	0.873
GOV2	Desc.	b	0.442	0.449	0.468	0.46	0.445	0.453	0.44
GOV2	Desc.	k_1	1.54	1.56	1.62	1.47	1.54	1.55	1.52
Clue-B	Titles	b	0.263	0.259	0.152	0.152	0.208	0.207	0.263
Clue-B	Titles	k_1	0.918	1.06	1.66	1.85	1.62	1.42	0.991
Clue-B	Desc.	b	0.367	0.285	0.38	0.287	0.331	0.33	0.335
Clue-B	Desc.	k_1	5.06	6.54	6.21	7.07	6.68	6.31	5.82

BM25-TP

The ranking function for the BM25-TP model (Rasolofo and Savoy, 2003) is defined as:

$$RSV_{BM25\text{-}TP}(D,Q) = RSV_{BM25}(D,Q) + \sum_{(t_i,t_{i+1})\in Q} w_D(t_i,t_{i+1}) \cdot \min(qw_i, qw_{i+1})$$

$$w_D(t_i,t_{i+1}) = \frac{(k_1+t_1)\cdot\sum_{occ_D(t_i,t_{i+1})} tpi(t_i,t_{i+1})}{K + \sum_{occ_D(t_i,t_{i+1})} tpi(t_i,t_{i+1})}$$

$$tpi(t_i,t_{i+1}) = \frac{1.0}{dist(t_i,t_{i+1})^2}$$

$$K = k_1 \cdot \left((1-b) + b \cdot \frac{|D|}{|D_{avg}|}\right)$$

This model combines the BM25 scoring function, defined above, with a score for each matched dependency. The parameters for the model are the same as for BM25, b and k_1. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Rob-04	Titles	b	0.296	0.337	0.222	0.223	0.225	0.261	0.219
Rob-04	Titles	k_1	0.552	0.701	0.792	0.791	0.748	0.717	0.763
Rob-04	Desc.	b	0.512	0.518	0.482	0.487	0.459	0.492	0.456
Rob-04	Desc.	k_1	0.868	0.842	0.879	0.758	0.89	0.847	0.954
GOV2	Titles	b	0.196	0.201	0.206	0.219	0.199	0.204	0.216
GOV2	Titles	k_1	0.668	0.727	0.627	0.773	0.793	0.718	0.724
GOV2	Desc.	b	0.337	0.333	0.351	0.313	0.328	0.332	0.334
GOV2	Desc.	k_1	1.1	1.16	1.35	1.12	1.14	1.17	1.17
Clue-B	Titles	b	0.18	0.184	0.1	0.1	0.13	0.139	0.119
Clue-B	Titles	k_1	1.98	1.95	1.99	1.98	2.36	2.05	1.83
Clue-B	Desc.	b	0.289	0.259	0.311	0.289	0.282	0.286	0.285
Clue-B	Desc.	k_1	4.29	5.81	4.35	5.7	4.27	4.88	4.98

BM25-TP2

The ranking function for the BM25-TP2 model (Svore et al., 2010) is defined as:

$$RSV_{BM25\text{-}TP2}(D,Q) = RSV_{BM25}(D,Q) + \sum_{(t_i,t_{i+1})\in Q} w_D(t_i,t_{i+1}) \cdot w_{i,i+1}$$

$$w_D(t_i,t_{i+1}) = \frac{(k_1+t_1)\cdot tf_{t_i,t_{i+1},D}}{K + tf_{t_i,t_{i+1},D}}$$

$$K = k_1 \cdot \left((1-b) + b \cdot \frac{|D|}{|D_{avg}|}\right)$$

where $tf_{t_i,t_{i+1},D}$ is the frequency of the bi-gram (t_i,t_{i+1}) in document D. Similar to BM25-TP, this model combines the BM25 scoring function, defined above, with a score for each matched dependency. And again, parameters for the model are the

same as for BM25, b and k_1. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Rob-04	Titles	b	0.213	0.238	0.155	0.179	0.135	0.184	0.184
Rob-04	Titles	k_1	0.668	0.918	0.982	0.728	1.28	0.914	0.914
Rob-04	Desc.	b	0.384	0.432	0.441	0.354	0.357	0.394	0.362
Rob-04	Desc.	k_1	1.39	1.43	1.28	1.21	1.27	1.32	1.27
GOV2	Titles	b	0.133	0.138	0.139	0.16	0.155	0.145	0.147
GOV2	Titles	k_1	0.782	1.16	1.17	0.71	1.33	1.03	1.26
GOV2	Desc.	b	0.183	0.174	0.206	0.192	0.18	0.187	0.182
GOV2	Desc.	k_1	1.42	2.03	1.47	1.34	1.42	1.54	1.43
Clue-B	Titles	b	0.109	0.0972	0.087	0.0902	0.129	0.102	0.11
Clue-B	Titles	k_1	2.98	5.04	3.52	3.06	4.85	3.89	5.21
Clue-B	Desc.	b	0.192	0.144	0.129	0.136	0.216	0.163	0.144
Clue-B	Desc.	k_1	4.99	6.64	9.32	7.1	5.62	6.73	6.45

BM25-Span

The ranking function for the BM25-Span model (Song et al., 2008) is defined as:

$$RSV_{Span}(Q,D) = \sum_{q_i \in Q} \log\left[\frac{N}{dc_{q_i}}\right] \cdot \frac{(k_1+1)\cdot rc}{k_1\left((1-b)+b\cdot\frac{|D|}{|D_{avg}|}\right)+rc}$$

$$rc = \sum_{occ_j \in spans(Q,D)} \frac{n_j^{\lambda}}{width(occ_j)^{\gamma}}$$

where $|D_{avg}|$ is the average length of a document in the collection, and dc_{q_i} is the number of documents that contain the query term, q_i. $spans(Q,D)$ is the function that returns all span occurrences, occ_j, in the document. n_j is the number of query terms present in the span instance occ_j. $width(occ_j)$ is the width of the span occ_j. The width of a singleton span is defined as the maximum width (45). The parameters for the model are b, k_1, γ and λ. The learned settings for these parameters for each collection and query set fold are listed in the following table.

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Rob-04	Titles	b	0.3	0.31	0.279	0.29	0.258	0.287	0.311
Rob-04	Titles	k_1	0.44	0.41	0.55	0.45	0.871	0.544	0.475
Rob-04	Titles	γ	0.25	0.22	0.227	0.24	0.145	0.216	0.236
Rob-04	Titles	λ	0.54	0.52	0.647	0.52	0.662	0.578	0.475
Rob-04	Desc.	b	0.46	0.5	0.479	0.535	0.531	0.501	0.501

Continued on next page

Continued from previous page

Coll.	Query Set	Param.	1	2	3	4	5	Avg.	Oracle
Rob-04	Desc.	k_1	0.74	0.57	0.866	0.695	0.554	0.685	0.685
Rob-04	Desc.	γ	0.17	0.25	0.17	0.175	0.263	0.206	0.206
Rob-04	Desc.	λ	0.44	0.4	0.494	0.354	0.538	0.445	0.465
GOV2	Titles	b	0.18	0.2	0.187	0.21	0.195	0.194	0.187
GOV2	Titles	k_1	0.584	0.574	0.653	0.615	0.598	0.605	0.644
GOV2	Titles	γ	0.312	0.312	0.315	0.295	0.337	0.314	0.316
GOV2	Titles	λ	1.335	1.305	1.308	1.212	1.208	1.274	1.308
GOV2	Desc.	b	0.307	0.331	0.341	0.321	0.347	0.33	0.331
GOV2	Desc.	k_1	1.054	1.335	1.025	1.005	1.204	1.125	1.325
GOV2	Desc.	γ	0.336	0.256	0.276	0.286	0.236	0.278	0.256
GOV2	Desc.	λ	0.878	0.745	0.655	0.695	0.638	0.722	0.725
Clue-B	Titles	b	0.27	0.15	0.074	0.12	0.135	0.15	0.12
Clue-B	Titles	k_1	3.254	0.75	3.074	1.05	1.093	1.844	1.914
Clue-B	Titles	γ	0.211	0.26	0.27	0.25	0.37	0.272	0.282
Clue-B	Titles	λ	1.487	1.16	1.538	1.2	1.141	1.305	1.305
Clue-B	Desc.	b	0.301	0.173	0.174	0.184	0.213	0.209	0.144
Clue-B	Desc.	k_1	3.885	4.561	5.885	10.855	4.921	6.021	6.425
Clue-B	Desc.	γ	0.206	0.334	0.284	0.314	0.43	0.314	0.324
Clue-B	Desc.	λ	0.735	1.1	1.074	1.254	1.29	1.09	1.284

Fold-level Retrieval Evaluation

In this section, we detail the evaluation metrics for each fold, retrieval model, collection, and query set. We also detail the evaluation metrics for the joint results (average of the evaluation metric over the 5 folds), the average parameter settings, and the oracle tuned set of parameters. We present retrieval models in the same order as above.

1. query likelihood (QL);
2. the sequential dependence model (SDM);
3. SDM variant: Uni+O234;
4. SDM variant: Uni+O234+U2;
5. SDM variant: Uni+O23+U23;
6. SDM variant: Uni+O234+U234;
7. the weighted sequential dependence model (WSDM);

8. WSDM variant: WSDM-Int;
9. WSDM variant: WSDM-Int-3;
10. PLM;
11. PLM-2;
12. PL2;
13. pDFR-BiL2;
14. pDFR-PL2;
15. BM25;
16. BM25-TP;
17. BM25-TP2; and
18. BM25-Span.

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
QL	Robust-04	Titles	0	0.261	0.402	0.361
QL	Robust-04	Titles	1	0.259	0.405	0.345
QL	Robust-04	Titles	2	0.259	0.443	0.402
QL	Robust-04	Titles	3	0.250	0.417	0.363
QL	Robust-04	Titles	4	0.231	0.394	0.356
QL	Robust-04	Titles	Joint	0.252	0.412	0.365
QL	Robust-04	Titles	Avg.	0.253	0.413	0.367
QL	Robust-04	Titles	Oracle	0.253	0.414	0.367
QL	Robust-04	Desc.	0	0.267	0.405	0.339
QL	Robust-04	Desc.	1	0.257	0.398	0.331
QL	Robust-04	Desc.	2	0.236	0.381	0.333
QL	Robust-04	Desc.	3	0.225	0.380	0.366
QL	Robust-04	Desc.	4	0.235	0.383	0.301
QL	Robust-04	Desc.	Joint	0.244	0.389	0.334
QL	Robust-04	Desc.	Avg.	0.244	0.389	0.333
QL	Robust-04	Desc.	Oracle	0.244	0.389	0.334
QL	GOV2	Titles	0	0.294	0.391	0.477
QL	GOV2	Titles	1	0.341	0.422	0.548
QL	GOV2	Titles	2	0.338	0.432	0.567
QL	GOV2	Titles	3	0.256	0.398	0.467
QL	GOV2	Titles	4	0.262	0.423	0.498
QL	GOV2	Titles	Joint	0.298	0.413	0.511
QL	GOV2	Titles	Avg.	0.298	0.415	0.512
QL	GOV2	Titles	Oracle	0.298	0.415	0.511
QL	GOV2	Desc.	0	0.210	0.329	0.374
QL	GOV2	Desc.	1	0.258	0.377	0.482
QL	GOV2	Desc.	2	0.258	0.409	0.437
QL	GOV2	Desc.	3	0.317	0.441	0.620
QL	GOV2	Desc.	4	0.240	0.333	0.447
QL	GOV2	Desc.	Joint	0.257	0.378	0.472
QL	GOV2	Desc.	Avg.	0.258	0.378	0.472
QL	GOV2	Desc.	Oracle	0.258	0.378	0.473
QL	ClueWeb-09-B	Titles	0	0.118	0.219	0.364
QL	ClueWeb-09-B	Titles	1	0.095	0.188	0.308
QL	ClueWeb-09-B	Titles	2	0.121	0.319	0.410
QL	ClueWeb-09-B	Titles	3	0.092	0.245	0.305
QL	ClueWeb-09-B	Titles	4	0.064	0.135	0.218
QL	ClueWeb-09-B	Titles	Joint	0.098	0.221	0.321
QL	ClueWeb-09-B	Titles	Avg.	0.099	0.224	0.326
QL	ClueWeb-09-B	Titles	Oracle	0.099	0.224	0.327
QL	ClueWeb-09-B	Desc.	0	0.048	0.154	0.213
QL	ClueWeb-09-B	Desc.	1	0.073	0.189	0.233
QL	ClueWeb-09-B	Desc.	2	0.107	0.223	0.264
QL	ClueWeb-09-B	Desc.	3	0.099	0.237	0.337
QL	ClueWeb-09-B	Desc.	4	0.045	0.143	0.172
QL	ClueWeb-09-B	Desc.	Joint	0.074	0.189	0.244
QL	ClueWeb-09-B	Desc.	Avg.	0.075	0.190	0.244
QL	ClueWeb-09-B	Desc.	Oracle	0.075	0.191	0.246
SDM	Robust-04	Titles	0	0.277	0.405	0.360
SDM	Robust-04	Titles	1	0.268	0.418	0.363

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
SDM	Robust-04	Titles	2	0.265	0.461	0.421
SDM	Robust-04	Titles	3	0.261	0.424	0.361
SDM	Robust-04	Titles	4	0.243	0.406	0.369
SDM	Robust-04	Titles	Joint	0.263	0.423	0.375
SDM	Robust-04	Titles	Avg.	0.264	0.423	0.375
SDM	Robust-04	Titles	Oracle	0.265	0.424	0.376
SDM	Robust-04	Desc.	0	0.276	0.413	0.350
SDM	Robust-04	Desc.	1	0.268	0.420	0.358
SDM	Robust-04	Desc.	2	0.256	0.400	0.342
SDM	Robust-04	Desc.	3	0.244	0.403	0.381
SDM	Robust-04	Desc.	4	0.246	0.395	0.315
SDM	Robust-04	Desc.	Joint	0.258	0.406	0.349
SDM	Robust-04	Desc.	Avg.	0.260	0.410	0.351
SDM	Robust-04	Desc.	Oracle	0.260	0.410	0.351
SDM	GOV2	Titles	0	0.327	0.430	0.510
SDM	GOV2	Titles	1	0.357	0.433	0.587
SDM	GOV2	Titles	2	0.361	0.479	0.626
SDM	GOV2	Titles	3	0.296	0.450	0.523
SDM	GOV2	Titles	4	0.290	0.452	0.543
SDM	GOV2	Titles	Joint	0.326	0.449	0.557
SDM	GOV2	Titles	Avg.	0.327	0.449	0.559
SDM	GOV2	Titles	Oracle	0.327	0.449	0.560
SDM	GOV2	Desc.	0	0.241	0.414	0.462
SDM	GOV2	Desc.	1	0.296	0.418	0.538
SDM	GOV2	Desc.	2	0.289	0.438	0.470
SDM	GOV2	Desc.	3	0.332	0.447	0.633
SDM	GOV2	Desc.	4	0.255	0.355	0.483
SDM	GOV2	Desc.	Joint	0.283	0.414	0.518
SDM	GOV2	Desc.	Avg.	0.284	0.414	0.517
SDM	GOV2	Desc.	Oracle	0.284	0.415	0.518
SDM	ClueWeb-09-B	Titles	0	0.128	0.266	0.412
SDM	ClueWeb-09-B	Titles	1	0.110	0.206	0.320
SDM	ClueWeb-09-B	Titles	2	0.125	0.322	0.415
SDM	ClueWeb-09-B	Titles	3	0.102	0.260	0.333
SDM	ClueWeb-09-B	Titles	4	0.075	0.140	0.238
SDM	ClueWeb-09-B	Titles	Joint	0.108	0.239	0.343
SDM	ClueWeb-09-B	Titles	Avg.	0.111	0.247	0.351
SDM	ClueWeb-09-B	Titles	Oracle	0.111	0.249	0.353
SDM	ClueWeb-09-B	Desc.	0	0.050	0.151	0.215
SDM	ClueWeb-09-B	Desc.	1	0.086	0.212	0.245
SDM	ClueWeb-09-B	Desc.	2	0.096	0.222	0.283
SDM	ClueWeb-09-B	Desc.	3	0.107	0.259	0.353
SDM	ClueWeb-09-B	Desc.	4	0.049	0.153	0.176
SDM	ClueWeb-09-B	Desc.	Joint	0.078	0.200	0.255
SDM	ClueWeb-09-B	Desc.	Avg.	0.080	0.200	0.254
SDM	ClueWeb-09-B	Desc.	Oracle	0.082	0.202	0.254
Uni+O234	Robust-04	Titles	0	0.279	0.416	0.372
Uni+O234	Robust-04	Titles	1	0.266	0.420	0.363
Uni+O234	Robust-04	Titles	2	0.266	0.462	0.419

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
Uni+O234	Robust-04	Titles	3	0.258	0.417	0.354
Uni+O234	Robust-04	Titles	4	0.238	0.394	0.356
Uni+O234	Robust-04	Titles	Joint	0.261	0.422	0.373
Uni+O234	Robust-04	Titles	Avg.	0.263	0.422	0.372
Uni+O234	Robust-04	Titles	Oracle	0.263	0.423	0.374
Uni+O234	Robust-04	Desc.	0	0.276	0.416	0.353
Uni+O234	Robust-04	Desc.	1	0.269	0.419	0.359
Uni+O234	Robust-04	Desc.	2	0.256	0.401	0.338
Uni+O234	Robust-04	Desc.	3	0.243	0.409	0.386
Uni+O234	Robust-04	Desc.	4	0.244	0.400	0.319
Uni+O234	Robust-04	Desc.	Joint	0.258	0.409	0.351
Uni+O234	Robust-04	Desc.	Avg.	0.258	0.411	0.354
Uni+O234	Robust-04	Desc.	Oracle	0.259	0.412	0.354
Uni+O234	GOV2	Titles	0	0.313	0.401	0.480
Uni+O234	GOV2	Titles	1	0.338	0.405	0.555
Uni+O234	GOV2	Titles	2	0.357	0.470	0.616
Uni+O234	GOV2	Titles	3	0.278	0.419	0.505
Uni+O234	GOV2	Titles	4	0.278	0.440	0.515
Uni+O234	GOV2	Titles	Joint	0.313	0.427	0.534
Uni+O234	GOV2	Titles	Avg.	0.316	0.430	0.533
Uni+O234	GOV2	Titles	Oracle	0.316	0.432	0.534
Uni+O234	GOV2	Desc.	0	0.236	0.397	0.448
Uni+O234	GOV2	Desc.	1	0.296	0.408	0.527
Uni+O234	GOV2	Desc.	2	0.285	0.429	0.477
Uni+O234	GOV2	Desc.	3	0.330	0.449	0.632
Uni+O234	GOV2	Desc.	4	0.247	0.352	0.472
Uni+O234	GOV2	Desc.	Joint	0.279	0.407	0.511
Uni+O234	GOV2	Desc.	Avg.	0.280	0.409	0.510
Uni+O234	GOV2	Desc.	Oracle	0.281	0.409	0.511
Uni+O234	ClueWeb-09-B	Titles	0	0.123	0.254	0.395
Uni+O234	ClueWeb-09-B	Titles	1	0.104	0.202	0.313
Uni+O234	ClueWeb-09-B	Titles	2	0.132	0.330	0.418
Uni+O234	ClueWeb-09-B	Titles	3	0.101	0.261	0.333
Uni+O234	ClueWeb-09-B	Titles	4	0.075	0.148	0.240
Uni+O234	ClueWeb-09-B	Titles	Joint	0.107	0.239	0.340
Uni+O234	ClueWeb-09-B	Titles	Avg.	0.110	0.245	0.347
Uni+O234	ClueWeb-09-B	Titles	Oracle	0.110	0.245	0.347
Uni+O234	ClueWeb-09-B	Desc.	0	0.048	0.155	0.223
Uni+O234	ClueWeb-09-B	Desc.	1	0.078	0.195	0.238
Uni+O234	ClueWeb-09-B	Desc.	2	0.092	0.205	0.248
Uni+O234	ClueWeb-09-B	Desc.	3	0.103	0.254	0.343
Uni+O234	ClueWeb-09-B	Desc.	4	0.048	0.138	0.169
Uni+O234	ClueWeb-09-B	Desc.	Joint	0.074	0.190	0.245
Uni+O234	ClueWeb-09-B	Desc.	Avg.	0.077	0.192	0.248
Uni+O234	ClueWeb-09-B	Desc.	Oracle	0.079	0.192	0.243
Uni+O234+U2	Robust-04	Titles	0	0.278	0.411	0.365
Uni+O234+U2	Robust-04	Titles	1	0.267	0.414	0.363
Uni+O234+U2	Robust-04	Titles	2	0.266	0.462	0.422
Uni+O234+U2	Robust-04	Titles	3	0.259	0.423	0.360

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
Uni+O234+U2	Robust-04	Titles	4	0.243	0.406	0.367
Uni+O234+U2	Robust-04	Titles	Joint	0.263	0.423	0.375
Uni+O234+U2	Robust-04	Titles	Avg.	0.264	0.424	0.376
Uni+O234+U2	Robust-04	Titles	Oracle	0.265	0.427	0.376
Uni+O234+U2	Robust-04	Desc.	0	0.277	0.415	0.348
Uni+O234+U2	Robust-04	Desc.	1	0.269	0.418	0.358
Uni+O234+U2	Robust-04	Desc.	2	0.258	0.406	0.346
Uni+O234+U2	Robust-04	Desc.	3	0.245	0.405	0.385
Uni+O234+U2	Robust-04	Desc.	4	0.247	0.398	0.317
Uni+O234+U2	Robust-04	Desc.	Joint	0.259	0.408	0.351
Uni+O234+U2	Robust-04	Desc.	Avg.	0.261	0.411	0.353
Uni+O234+U2	Robust-04	Desc.	Oracle	0.261	0.413	0.354
Uni+O234+U2	GOV2	Titles	0	0.328	0.432	0.517
Uni+O234+U2	GOV2	Titles	1	0.353	0.425	0.578
Uni+O234+U2	GOV2	Titles	2	0.360	0.478	0.622
Uni+O234+U2	GOV2	Titles	3	0.294	0.442	0.517
Uni+O234+U2	GOV2	Titles	4	0.290	0.454	0.545
Uni+O234+U2	GOV2	Titles	Joint	0.325	0.446	0.555
Uni+O234+U2	GOV2	Titles	Avg.	0.328	0.449	0.557
Uni+O234+U2	GOV2	Titles	Oracle	0.328	0.450	0.558
Uni+O234+U2	GOV2	Desc.	0	0.241	0.414	0.464
Uni+O234+U2	GOV2	Desc.	1	0.297	0.419	0.540
Uni+O234+U2	GOV2	Desc.	2	0.288	0.428	0.475
Uni+O234+U2	GOV2	Desc.	3	0.332	0.439	0.622
Uni+O234+U2	GOV2	Desc.	4	0.257	0.343	0.477
Uni+O234+U2	GOV2	Desc.	Joint	0.283	0.409	0.516
Uni+O234+U2	GOV2	Desc.	Avg.	0.284	0.410	0.515
Uni+O234+U2	GOV2	Desc.	Oracle	0.284	0.412	0.516
Uni+O234+U2	ClueWeb-09-B	Titles	0	0.129	0.280	0.428
Uni+O234+U2	ClueWeb-09-B	Titles	1	0.108	0.218	0.337
Uni+O234+U2	ClueWeb-09-B	Titles	2	0.128	0.326	0.422
Uni+O234+U2	ClueWeb-09-B	Titles	3	0.100	0.255	0.326
Uni+O234+U2	ClueWeb-09-B	Titles	4	0.078	0.138	0.238
Uni+O234+U2	ClueWeb-09-B	Titles	Joint	0.109	0.243	0.350
Uni+O234+U2	ClueWeb-09-B	Titles	Avg.	0.112	0.248	0.353
Uni+O234+U2	ClueWeb-09-B	Titles	Oracle	0.112	0.247	0.351
Uni+O234+U2	ClueWeb-09-B	Desc.	0	0.049	0.148	0.212
Uni+O234+U2	ClueWeb-09-B	Desc.	1	0.088	0.216	0.253
Uni+O234+U2	ClueWeb-09-B	Desc.	2	0.105	0.228	0.279
Uni+O234+U2	ClueWeb-09-B	Desc.	3	0.106	0.265	0.357
Uni+O234+U2	ClueWeb-09-B	Desc.	4	0.050	0.153	0.179
Uni+O234+U2	ClueWeb-09-B	Desc.	Joint	0.079	0.202	0.256
Uni+O234+U2	ClueWeb-09-B	Desc.	Avg.	0.082	0.201	0.255
Uni+O234+U2	ClueWeb-09-B	Desc.	Oracle	0.082	0.203	0.256
Uni+O23+U23	Robust-04	Titles	0	0.276	0.415	0.369
Uni+O23+U23	Robust-04	Titles	1	0.269	0.417	0.365
Uni+O23+U23	Robust-04	Titles	2	0.267	0.461	0.416
Uni+O23+U23	Robust-04	Titles	3	0.262	0.430	0.369
Uni+O23+U23	Robust-04	Titles	4	0.242	0.399	0.357

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
Uni+O23+U23	Robust-04	Titles	Joint	0.263	0.424	0.375
Uni+O23+U23	Robust-04	Titles	Avg.	0.265	0.426	0.377
Uni+O23+U23	Robust-04	Titles	Oracle	0.265	0.427	0.379
Uni+O23+U23	Robust-04	Desc.	0	0.277	0.418	0.354
Uni+O23+U23	Robust-04	Desc.	1	0.270	0.424	0.357
Uni+O23+U23	Robust-04	Desc.	2	0.258	0.404	0.342
Uni+O23+U23	Robust-04	Desc.	3	0.244	0.404	0.382
Uni+O23+U23	Robust-04	Desc.	4	0.249	0.398	0.315
Uni+O23+U23	Robust-04	Desc.	Joint	0.260	0.410	0.350
Uni+O23+U23	Robust-04	Desc.	Avg.	0.261	0.412	0.352
Uni+O23+U23	Robust-04	Desc.	Oracle	0.261	0.410	0.352
Uni+O23+U23	GOV2	Titles	0	0.322	0.423	0.507
Uni+O23+U23	GOV2	Titles	1	0.357	0.431	0.580
Uni+O23+U23	GOV2	Titles	2	0.358	0.467	0.607
Uni+O23+U23	GOV2	Titles	3	0.296	0.451	0.530
Uni+O23+U23	GOV2	Titles	4	0.295	0.455	0.545
Uni+O23+U23	GOV2	Titles	Joint	0.325	0.445	0.553
Uni+O23+U23	GOV2	Titles	Avg.	0.329	0.448	0.555
Uni+O23+U23	GOV2	Titles	Oracle	0.327	0.449	0.555
Uni+O23+U23	GOV2	Desc.	0	0.238	0.401	0.453
Uni+O23+U23	GOV2	Desc.	1	0.292	0.412	0.528
Uni+O23+U23	GOV2	Desc.	2	0.287	0.430	0.487
Uni+O23+U23	GOV2	Desc.	3	0.332	0.444	0.628
Uni+O23+U23	GOV2	Desc.	4	0.255	0.356	0.483
Uni+O23+U23	GOV2	Desc.	Joint	0.281	0.409	0.516
Uni+O23+U23	GOV2	Desc.	Avg.	0.285	0.412	0.519
Uni+O23+U23	GOV2	Desc.	Oracle	0.283	0.410	0.513
Uni+O23+U23	ClueWeb-09-B	Titles	0	0.130	0.279	0.429
Uni+O23+U23	ClueWeb-09-B	Titles	1	0.107	0.215	0.333
Uni+O23+U23	ClueWeb-09-B	Titles	2	0.127	0.330	0.430
Uni+O23+U23	ClueWeb-09-B	Titles	3	0.099	0.255	0.324
Uni+O23+U23	ClueWeb-09-B	Titles	4	0.077	0.136	0.232
Uni+O23+U23	ClueWeb-09-B	Titles	Joint	0.108	0.243	0.349
Uni+O23+U23	ClueWeb-09-B	Titles	Avg.	0.112	0.252	0.355
Uni+O23+U23	ClueWeb-09-B	Titles	Oracle	0.112	0.251	0.352
Uni+O23+U23	ClueWeb-09-B	Desc.	0	0.052	0.156	0.228
Uni+O23+U23	ClueWeb-09-B	Desc.	1	0.081	0.208	0.243
Uni+O23+U23	ClueWeb-09-B	Desc.	2	0.094	0.221	0.283
Uni+O23+U23	ClueWeb-09-B	Desc.	3	0.106	0.262	0.343
Uni+O23+U23	ClueWeb-09-B	Desc.	4	0.045	0.129	0.162
Uni+O23+U23	ClueWeb-09-B	Desc.	Joint	0.076	0.196	0.252
Uni+O23+U23	ClueWeb-09-B	Desc.	Avg.	0.081	0.201	0.250
Uni+O23+U23	ClueWeb-09-B	Desc.	Oracle	0.078	0.199	0.255
Uni+O234+U234	Robust-04	Desc.	0	0.277	0.419	0.355
Uni+O234+U234	Robust-04	Desc.	1	0.267	0.420	0.360
Uni+O234+U234	Robust-04	Desc.	2	0.259	0.398	0.338
Uni+O234+U234	Robust-04	Desc.	3	0.244	0.408	0.387
Uni+O234+U234	Robust-04	Desc.	4	0.249	0.402	0.324
Uni+O234+U234	Robust-04	Desc.	Joint	0.259	0.409	0.353

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
Uni+O234+U234	Robust-04	Desc.	Avg.	0.261	0.412	0.353
Uni+O234+U234	Robust-04	Desc.	Oracle	0.260	0.411	0.352
Uni+O234+U234	GOV2	Desc.	0	0.240	0.410	0.460
Uni+O234+U234	GOV2	Desc.	1	0.293	0.422	0.540
Uni+O234+U234	GOV2	Desc.	2	0.283	0.424	0.467
Uni+O234+U234	GOV2	Desc.	3	0.333	0.447	0.623
Uni+O234+U234	GOV2	Desc.	4	0.256	0.348	0.463
Uni+O234+U234	GOV2	Desc.	Joint	0.282	0.410	0.511
Uni+O234+U234	GOV2	Desc.	Avg.	0.285	0.411	0.518
Uni+O234+U234	GOV2	Desc.	Oracle	0.284	0.406	0.518
Uni+O234+U234	ClueWeb-09-B	Desc.	0	0.051	0.157	0.225
Uni+O234+U234	ClueWeb-09-B	Desc.	1	0.086	0.206	0.232
Uni+O234+U234	ClueWeb-09-B	Desc.	2	0.105	0.225	0.264
Uni+O234+U234	ClueWeb-09-B	Desc.	3	0.106	0.259	0.357
Uni+O234+U234	ClueWeb-09-B	Desc.	4	0.049	0.148	0.184
Uni+O234+U234	ClueWeb-09-B	Desc.	Joint	0.079	0.199	0.253
Uni+O234+U234	ClueWeb-09-B	Desc.	Avg.	0.082	0.203	0.254
Uni+O234+U234	ClueWeb-09-B	Desc.	Oracle	0.076	0.198	0.257
WSDM	Robust-04	Titles	0	0.292	0.427	0.377
WSDM	Robust-04	Titles	1	0.275	0.428	0.368
WSDM	Robust-04	Titles	2	0.277	0.470	0.428
WSDM	Robust-04	Titles	3	0.269	0.442	0.382
WSDM	Robust-04	Titles	4	0.244	0.406	0.363
WSDM	Robust-04	Titles	Joint	0.271	0.435	0.383
WSDM	Robust-04	Titles	Avg.	0.272	0.435	0.383
WSDM	Robust-04	Titles	Oracle	0.272	0.436	0.384
WSDM	Robust-04	Desc.	0	0.297	0.429	0.362
WSDM	Robust-04	Desc.	1	0.289	0.433	0.365
WSDM	Robust-04	Desc.	2	0.286	0.438	0.358
WSDM	Robust-04	Desc.	3	0.263	0.426	0.401
WSDM	Robust-04	Desc.	4	0.278	0.431	0.342
WSDM	Robust-04	Desc.	Joint	0.283	0.431	0.366
WSDM	Robust-04	Desc.	Avg.	0.283	0.434	0.370
WSDM	Robust-04	Desc.	Oracle	0.284	0.433	0.368
WSDM	GOV2	Titles	0	0.331	0.419	0.497
WSDM	GOV2	Titles	1	0.358	0.438	0.592
WSDM	GOV2	Titles	2	0.365	0.482	0.641
WSDM	GOV2	Titles	3	0.295	0.451	0.527
WSDM	GOV2	Titles	4	0.307	0.476	0.563
WSDM	GOV2	Titles	Joint	0.331	0.453	0.563
WSDM	GOV2	Titles	Avg.	0.333	0.456	0.565
WSDM	GOV2	Titles	Oracle	0.333	0.453	0.562
WSDM	GOV2	Desc.	0	0.277	0.441	0.519
WSDM	GOV2	Desc.	1	0.331	0.439	0.555
WSDM	GOV2	Desc.	2	0.303	0.459	0.497
WSDM	GOV2	Desc.	3	0.352	0.456	0.653
WSDM	GOV2	Desc.	4	0.253	0.334	0.457
WSDM	GOV2	Desc.	Joint	0.303	0.426	0.536
WSDM	GOV2	Desc.	Avg.	0.308	0.431	0.540

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
WSDM	GOV2	Desc.	Oracle	0.305	0.439	0.544
WSDM	ClueWeb-09-B	Titles	0	0.129	0.265	0.416
WSDM	ClueWeb-09-B	Titles	1	0.116	0.230	0.352
WSDM	ClueWeb-09-B	Titles	2	0.134	0.327	0.418
WSDM	ClueWeb-09-B	Titles	3	0.102	0.255	0.331
WSDM	ClueWeb-09-B	Titles	4	0.077	0.143	0.243
WSDM	ClueWeb-09-B	Titles	Joint	0.111	0.244	0.352
WSDM	ClueWeb-09-B	Titles	Avg.	0.114	0.251	0.359
WSDM	ClueWeb-09-B	Titles	Oracle	0.114	0.250	0.360
WSDM	ClueWeb-09-B	Desc.	0	0.062	0.182	0.257
WSDM	ClueWeb-09-B	Desc.	1	0.092	0.235	0.270
WSDM	ClueWeb-09-B	Desc.	2	0.119	0.235	0.274
WSDM	ClueWeb-09-B	Desc.	3	0.111	0.279	0.388
WSDM	ClueWeb-09-B	Desc.	4	0.057	0.161	0.222
WSDM	ClueWeb-09-B	Desc.	Joint	0.088	0.219	0.283
WSDM	ClueWeb-09-B	Desc.	Avg.	0.089	0.218	0.287
WSDM	ClueWeb-09-B	Desc.	Oracle	0.088	0.220	0.284
WSDM-Int	Robust-04	Titles	0	0.288	0.427	0.379
WSDM-Int	Robust-04	Titles	1	0.274	0.428	0.368
WSDM-Int	Robust-04	Titles	2	0.278	0.473	0.432
WSDM-Int	Robust-04	Titles	3	0.265	0.436	0.373
WSDM-Int	Robust-04	Titles	4	0.241	0.396	0.356
WSDM-Int	Robust-04	Titles	Joint	0.269	0.432	0.382
WSDM-Int	Robust-04	Titles	Avg.	0.272	0.434	0.382
WSDM-Int	Robust-04	Titles	Oracle	0.272	0.436	0.384
WSDM-Int	Robust-04	Desc.	0	0.294	0.431	0.361
WSDM-Int	Robust-04	Desc.	1	0.286	0.429	0.363
WSDM-Int	Robust-04	Desc.	2	0.276	0.431	0.359
WSDM-Int	Robust-04	Desc.	3	0.259	0.417	0.391
WSDM-Int	Robust-04	Desc.	4	0.274	0.430	0.350
WSDM-Int	Robust-04	Desc.	Joint	0.278	0.428	0.365
WSDM-Int	Robust-04	Desc.	Avg.	0.281	0.430	0.365
WSDM-Int	Robust-04	Desc.	Oracle	0.281	0.431	0.367
WSDM-Int	GOV2	Titles	0	0.330	0.418	0.487
WSDM-Int	GOV2	Titles	1	0.355	0.435	0.593
WSDM-Int	GOV2	Titles	2	0.363	0.481	0.636
WSDM-Int	GOV2	Titles	3	0.292	0.443	0.517
WSDM-Int	GOV2	Titles	4	0.306	0.473	0.552
WSDM-Int	GOV2	Titles	Joint	0.329	0.450	0.556
WSDM-Int	GOV2	Titles	Avg.	0.333	0.455	0.563
WSDM-Int	GOV2	Titles	Oracle	0.333	0.455	0.563
WSDM-Int	GOV2	Desc.	0	0.270	0.448	0.524
WSDM-Int	GOV2	Desc.	1	0.321	0.431	0.545
WSDM-Int	GOV2	Desc.	2	0.294	0.455	0.488
WSDM-Int	GOV2	Desc.	3	0.347	0.452	0.648
WSDM-Int	GOV2	Desc.	4	0.257	0.341	0.458
WSDM-Int	GOV2	Desc.	Joint	0.298	0.425	0.533
WSDM-Int	GOV2	Desc.	Avg.	0.301	0.428	0.536
WSDM-Int	GOV2	Desc.	Oracle	0.300	0.423	0.534

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
WSDM-Int	ClueWeb-09-B	Titles	0	0.129	0.264	0.417
WSDM-Int	ClueWeb-09-B	Titles	1	0.116	0.230	0.352
WSDM-Int	ClueWeb-09-B	Titles	2	0.140	0.335	0.430
WSDM-Int	ClueWeb-09-B	Titles	3	0.102	0.258	0.329
WSDM-Int	ClueWeb-09-B	Titles	4	0.075	0.139	0.245
WSDM-Int	ClueWeb-09-B	Titles	Joint	0.113	0.245	0.354
WSDM-Int	ClueWeb-09-B	Titles	Avg.	0.114	0.250	0.360
WSDM-Int	ClueWeb-09-B	Titles	Oracle	0.114	0.248	0.357
WSDM-Int	ClueWeb-09-B	Desc.	0	0.053	0.147	0.220
WSDM-Int	ClueWeb-09-B	Desc.	1	0.089	0.214	0.255
WSDM-Int	ClueWeb-09-B	Desc.	2	0.110	0.216	0.236
WSDM-Int	ClueWeb-09-B	Desc.	3	0.107	0.263	0.367
WSDM-Int	ClueWeb-09-B	Desc.	4	0.053	0.152	0.195
WSDM-Int	ClueWeb-09-B	Desc.	Joint	0.083	0.199	0.255
WSDM-Int	ClueWeb-09-B	Desc.	Avg.	0.086	0.209	0.270
WSDM-Int	ClueWeb-09-B	Desc.	Oracle	0.086	0.211	0.271
WSDM-Int-3	Robust-04	Titles	0	0.291	0.434	0.382
WSDM-Int-3	Robust-04	Titles	1	0.274	0.426	0.365
WSDM-Int-3	Robust-04	Titles	2	0.279	0.472	0.430
WSDM-Int-3	Robust-04	Titles	3	0.269	0.448	0.386
WSDM-Int-3	Robust-04	Titles	4	0.243	0.404	0.362
WSDM-Int-3	Robust-04	Titles	Joint	0.271	0.437	0.385
WSDM-Int-3	Robust-04	Titles	Avg.	0.273	0.437	0.384
WSDM-Int-3	Robust-04	Titles	Oracle	0.273	0.437	0.385
WSDM-Int-3	Robust-04	Desc.	0	0.295	0.431	0.357
WSDM-Int-3	Robust-04	Desc.	1	0.287	0.435	0.374
WSDM-Int-3	Robust-04	Desc.	2	0.281	0.432	0.359
WSDM-Int-3	Robust-04	Desc.	3	0.261	0.419	0.392
WSDM-Int-3	Robust-04	Desc.	4	0.275	0.424	0.338
WSDM-Int-3	Robust-04	Desc.	Joint	0.280	0.428	0.364
WSDM-Int-3	Robust-04	Desc.	Avg.	0.282	0.430	0.365
WSDM-Int-3	Robust-04	Desc.	Oracle	0.282	0.431	0.366
WSDM-Int-3	GOV2	Titles	0	0.334	0.419	0.490
WSDM-Int-3	GOV2	Titles	1	0.358	0.436	0.588
WSDM-Int-3	GOV2	Titles	2	0.362	0.480	0.645
WSDM-Int-3	GOV2	Titles	3	0.293	0.441	0.508
WSDM-Int-3	GOV2	Titles	4	0.304	0.468	0.550
WSDM-Int-3	GOV2	Titles	Joint	0.330	0.448	0.556
WSDM-Int-3	GOV2	Titles	Avg.	0.333	0.456	0.563
WSDM-Int-3	GOV2	Titles	Oracle	0.334	0.455	0.560
WSDM-Int-3	GOV2	Desc.	0	0.270	0.448	0.521
WSDM-Int-3	GOV2	Desc.	1	0.321	0.430	0.545
WSDM-Int-3	GOV2	Desc.	2	0.295	0.453	0.483
WSDM-Int-3	GOV2	Desc.	3	0.343	0.452	0.650
WSDM-Int-3	GOV2	Desc.	4	0.256	0.341	0.457
WSDM-Int-3	GOV2	Desc.	Joint	0.297	0.425	0.531
WSDM-Int-3	GOV2	Desc.	Avg.	0.301	0.427	0.534
WSDM-Int-3	GOV2	Desc.	Oracle	0.302	0.427	0.535
WSDM-Int-3	ClueWeb-09-B	Titles	0	0.130	0.266	0.414

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
WSDM-Int-3	ClueWeb-09-B	Titles	1	0.108	0.208	0.317
WSDM-Int-3	ClueWeb-09-B	Titles	2	0.137	0.337	0.432
WSDM-Int-3	ClueWeb-09-B	Titles	3	0.099	0.254	0.326
WSDM-Int-3	ClueWeb-09-B	Titles	4	0.081	0.149	0.247
WSDM-Int-3	ClueWeb-09-B	Titles	Joint	0.111	0.242	0.347
WSDM-Int-3	ClueWeb-09-B	Titles	Avg.	0.114	0.250	0.360
WSDM-Int-3	ClueWeb-09-B	Titles	Oracle	0.114	0.246	0.353
WSDM-Int-3	ClueWeb-09-B	Desc.	0	0.055	0.163	0.212
WSDM-Int-3	ClueWeb-09-B	Desc.	1	0.083	0.213	0.248
WSDM-Int-3	ClueWeb-09-B	Desc.	2	0.103	0.215	0.245
WSDM-Int-3	ClueWeb-09-B	Desc.	3	0.105	0.265	0.367
WSDM-Int-3	ClueWeb-09-B	Desc.	4	0.053	0.154	0.184
WSDM-Int-3	ClueWeb-09-B	Desc.	Joint	0.080	0.202	0.252
WSDM-Int-3	ClueWeb-09-B	Desc.	Avg.	0.086	0.210	0.262
WSDM-Int-3	ClueWeb-09-B	Desc.	Oracle	0.086	0.212	0.267
PLM	Robust-04	Titles	0	0.256	0.393	0.353
PLM	Robust-04	Titles	1	0.261	0.409	0.348
PLM	Robust-04	Titles	2	0.257	0.440	0.403
PLM	Robust-04	Titles	3	0.252	0.418	0.365
PLM	Robust-04	Titles	4	0.234	0.393	0.352
PLM	Robust-04	Titles	Joint	0.252	0.411	0.364
PLM	Robust-04	Titles	Avg.	0.253	0.412	0.365
PLM	Robust-04	Titles	Oracle	0.254	0.412	0.365
PLM	Robust-04	Desc.	0	0.269	0.389	0.328
PLM	Robust-04	Desc.	1	0.253	0.382	0.322
PLM	Robust-04	Desc.	2	0.248	0.385	0.344
PLM	Robust-04	Desc.	3	0.235	0.395	0.364
PLM	Robust-04	Desc.	4	0.244	0.382	0.300
PLM	Robust-04	Desc.	Joint	0.250	0.386	0.332
PLM	Robust-04	Desc.	Avg.	0.251	0.387	0.332
PLM	Robust-04	Desc.	Oracle	0.251	0.387	0.332
PLM	GOV2	Titles	Joint	0.305	0.406	0.509
PLM	GOV2	Desc.	Joint	0.247	0.337	0.433
PLM	ClueWeb-09-B	Titles	Joint	0.092	0.207	0.301
PLM	ClueWeb-09-B	Desc.	Joint	0.064	0.153	0.198
PLM-2	Robust-04	Titles	0	0.264	0.396	0.352
PLM-2	Robust-04	Titles	1	0.263	0.407	0.352
PLM-2	Robust-04	Titles	2	0.264	0.451	0.406
PLM-2	Robust-04	Titles	3	0.256	0.418	0.368
PLM-2	Robust-04	Titles	4	0.234	0.398	0.361
PLM-2	Robust-04	Titles	Joint	0.256	0.414	0.368
PLM-2	Robust-04	Titles	Avg.	0.248	0.409	0.359
PLM-2	Robust-04	Titles	Oracle	0.257	0.415	0.367
PLM-2	Robust-04	Desc.	0	0.282	0.413	0.348
PLM-2	Robust-04	Desc.	1	0.265	0.404	0.343
PLM-2	Robust-04	Desc.	2	0.254	0.385	0.339
PLM-2	Robust-04	Desc.	3	0.241	0.394	0.378
PLM-2	Robust-04	Desc.	4	0.255	0.395	0.318
PLM-2	Robust-04	Desc.	Joint	0.260	0.398	0.345

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
PLM-2	Robust-04	Desc.	Avg.	0.256	0.395	0.344
PLM-2	Robust-04	Desc.	Oracle	0.259	0.398	0.346
PLM-2	GOV2	Titles	Avg.†	0.302	0.410	0.506
PLM-2	GOV2	Desc.	Avg.†	0.276	0.390	0.485
PLM-2	ClueWeb-09-B	Titles	Avg.†	0.097	0.214	0.306
PLM-2	ClueWeb-09-B	Desc.	Avg.†	0.075	0.190	0.239
PL2	Robust-04	Titles	0	0.263	0.405	0.362
PL2	Robust-04	Titles	1	0.263	0.413	0.349
PL2	Robust-04	Titles	2	0.258	0.452	0.403
PL2	Robust-04	Titles	3	0.252	0.419	0.367
PL2	Robust-04	Titles	4	0.230	0.402	0.363
PL2	Robust-04	Titles	Joint	0.253	0.418	0.369
PL2	Robust-04	Titles	Avg.	0.253	0.418	0.368
PL2	Robust-04	Titles	Oracle	0.253	0.418	0.369
PL2	Robust-04	Desc.	0	0.247	0.392	0.327
PL2	Robust-04	Desc.	1	0.245	0.390	0.330
PL2	Robust-04	Desc.	2	0.228	0.398	0.333
PL2	Robust-04	Desc.	3	0.210	0.396	0.359
PL2	Robust-04	Desc.	4	0.217	0.367	0.296
PL2	Robust-04	Desc.	Joint	0.229	0.389	0.329
PL2	Robust-04	Desc.	Avg.	0.230	0.388	0.328
PL2	Robust-04	Desc.	Oracle	0.230	0.389	0.329
PL2	GOV2	Titles	0	0.302	0.413	0.507
PL2	GOV2	Titles	1	0.337	0.423	0.547
PL2	GOV2	Titles	2	0.341	0.423	0.559
PL2	GOV2	Titles	3	0.261	0.398	0.467
PL2	GOV2	Titles	4	0.258	0.417	0.503
PL2	GOV2	Titles	Joint	0.300	0.415	0.516
PL2	GOV2	Titles	Avg.	0.300	0.415	0.516
PL2	GOV2	Titles	Oracle	0.301	0.415	0.517
PL2	GOV2	Desc.	0	0.208	0.364	0.409
PL2	GOV2	Desc.	1	0.251	0.402	0.497
PL2	GOV2	Desc.	2	0.262	0.426	0.437
PL2	GOV2	Desc.	3	0.325	0.432	0.615
PL2	GOV2	Desc.	4	0.244	0.324	0.435
PL2	GOV2	Desc.	Joint	0.258	0.390	0.479
PL2	GOV2	Desc.	Avg.	0.259	0.391	0.478
PL2	GOV2	Desc.	Oracle	0.259	0.391	0.478
PL2	ClueWeb-09-B	Titles	0	0.123	0.238	0.397
PL2	ClueWeb-09-B	Titles	1	0.110	0.211	0.328
PL2	ClueWeb-09-B	Titles	2	0.128	0.330	0.425
PL2	ClueWeb-09-B	Titles	3	0.094	0.249	0.310
PL2	ClueWeb-09-B	Titles	4	0.070	0.136	0.228
PL2	ClueWeb-09-B	Titles	Joint	0.105	0.233	0.337
PL2	ClueWeb-09-B	Titles	Avg.	0.105	0.233	0.337
PL2	ClueWeb-09-B	Titles	Oracle	0.105	0.233	0.337
PL2	ClueWeb-09-B	Desc.	0	0.049	0.150	0.198
PL2	ClueWeb-09-B	Desc.	1	0.076	0.192	0.223
PL2	ClueWeb-09-B	Desc.	2	0.107	0.221	0.260

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
PL2	ClueWeb-09-B	Desc.	3	0.106	0.257	0.358
PL2	ClueWeb-09-B	Desc.	4	0.046	0.148	0.193
PL2	ClueWeb-09-B	Desc.	Joint	0.077	0.194	0.247
PL2	ClueWeb-09-B	Desc.	Avg.	0.078	0.194	0.249
PL2	ClueWeb-09-B	Desc.	Oracle	0.078	0.193	0.247
pDFR-BiL2	Robust-04	Titles	0	0.267	0.402	0.363
pDFR-BiL2	Robust-04	Titles	1	0.271	0.422	0.366
pDFR-BiL2	Robust-04	Titles	2	0.262	0.459	0.409
pDFR-BiL2	Robust-04	Titles	3	0.255	0.418	0.358
pDFR-BiL2	Robust-04	Titles	4	0.237	0.409	0.365
pDFR-BiL2	Robust-04	Titles	Joint	0.258	0.422	0.372
pDFR-BiL2	Robust-04	Titles	Avg.	0.259	0.422	0.372
pDFR-BiL2	Robust-04	Titles	Oracle	0.260	0.424	0.374
pDFR-BiL2	Robust-04	Desc.	0	0.255	0.402	0.335
pDFR-BiL2	Robust-04	Desc.	1	0.251	0.405	0.343
pDFR-BiL2	Robust-04	Desc.	2	0.234	0.394	0.330
pDFR-BiL2	Robust-04	Desc.	3	0.207	0.388	0.361
pDFR-BiL2	Robust-04	Desc.	4	0.222	0.378	0.306
pDFR-BiL2	Robust-04	Desc.	Joint	0.234	0.393	0.335
pDFR-BiL2	Robust-04	Desc.	Avg.	0.235	0.394	0.335
pDFR-BiL2	Robust-04	Desc.	Oracle	0.236	0.397	0.336
pDFR-BiL2	GOV2	Titles	0	0.320	0.434	0.533
pDFR-BiL2	GOV2	Titles	1	0.353	0.447	0.580
pDFR-BiL2	GOV2	Titles	2	0.349	0.463	0.600
pDFR-BiL2	GOV2	Titles	3	0.277	0.411	0.473
pDFR-BiL2	GOV2	Titles	4	0.267	0.430	0.495
pDFR-BiL2	GOV2	Titles	Joint	0.313	0.437	0.536
pDFR-BiL2	GOV2	Titles	Avg.	0.314	0.436	0.536
pDFR-BiL2	GOV2	Titles	Oracle	0.314	0.436	0.536
pDFR-BiL2	GOV2	Desc.	0	0.217	0.363	0.410
pDFR-BiL2	GOV2	Desc.	1	0.255	0.399	0.495
pDFR-BiL2	GOV2	Desc.	2	0.269	0.445	0.467
pDFR-BiL2	GOV2	Desc.	3	0.334	0.437	0.623
pDFR-BiL2	GOV2	Desc.	4	0.252	0.325	0.433
pDFR-BiL2	GOV2	Desc.	Joint	0.266	0.394	0.486
pDFR-BiL2	GOV2	Desc.	Avg.	0.267	0.394	0.484
pDFR-BiL2	GOV2	Desc.	Oracle	0.267	0.395	0.484
pDFR-BiL2	ClueWeb-09-B	Titles	0	0.124	0.232	0.384
pDFR-BiL2	ClueWeb-09-B	Titles	1	0.100	0.189	0.298
pDFR-BiL2	ClueWeb-09-B	Titles	2	0.124	0.318	0.407
pDFR-BiL2	ClueWeb-09-B	Titles	3	0.093	0.254	0.314
pDFR-BiL2	ClueWeb-09-B	Titles	4	0.070	0.135	0.228
pDFR-BiL2	ClueWeb-09-B	Titles	Joint	0.102	0.225	0.326
pDFR-BiL2	ClueWeb-09-B	Titles	Avg.	0.105	0.232	0.339
pDFR-BiL2	ClueWeb-09-B	Titles	Oracle	0.106	0.230	0.330
pDFR-BiL2	ClueWeb-09-B	Desc.	0	0.048	0.151	0.200
pDFR-BiL2	ClueWeb-09-B	Desc.	1	0.076	0.190	0.222
pDFR-BiL2	ClueWeb-09-B	Desc.	2	0.106	0.214	0.248
pDFR-BiL2	ClueWeb-09-B	Desc.	3	0.104	0.252	0.355

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
pDFR-BiL2	ClueWeb-09-B	Desc.	4	0.046	0.150	0.197
pDFR-BiL2	ClueWeb-09-B	Desc.	Joint	0.076	0.192	0.245
pDFR-BiL2	ClueWeb-09-B	Desc.	Avg.	0.078	0.194	0.247
pDFR-BiL2	ClueWeb-09-B	Desc.	Oracle	0.078	0.193	0.247
pDFR-PL2	Robust-04	Titles	0	0.274	0.406	0.365
pDFR-PL2	Robust-04	Titles	1	0.270	0.424	0.365
pDFR-PL2	Robust-04	Titles	2	0.259	0.450	0.411
pDFR-PL2	Robust-04	Titles	3	0.258	0.423	0.364
pDFR-PL2	Robust-04	Titles	4	0.238	0.408	0.370
pDFR-PL2	Robust-04	Titles	Joint	0.260	0.422	0.375
pDFR-PL2	Robust-04	Titles	Avg.	0.261	0.423	0.376
pDFR-PL2	Robust-04	Titles	Oracle	0.261	0.423	0.376
pDFR-PL2	Robust-04	Desc.	0	0.256	0.403	0.333
pDFR-PL2	Robust-04	Desc.	1	0.253	0.399	0.338
pDFR-PL2	Robust-04	Desc.	2	0.234	0.396	0.334
pDFR-PL2	Robust-04	Desc.	3	0.210	0.392	0.357
pDFR-PL2	Robust-04	Desc.	4	0.225	0.374	0.300
pDFR-PL2	Robust-04	Desc.	Joint	0.235	0.393	0.333
pDFR-PL2	Robust-04	Desc.	Avg.	0.237	0.396	0.337
pDFR-PL2	Robust-04	Desc.	Oracle	0.237	0.398	0.337
pDFR-PL2	GOV2	Titles	0	0.324	0.436	0.528
pDFR-PL2	GOV2	Titles	1	0.353	0.439	0.587
pDFR-PL2	GOV2	Titles	2	0.339	0.447	0.581
pDFR-PL2	GOV2	Titles	3	0.286	0.431	0.492
pDFR-PL2	GOV2	Titles	4	0.286	0.452	0.533
pDFR-PL2	GOV2	Titles	Joint	0.317	0.441	0.544
pDFR-PL2	GOV2	Titles	Avg.	0.321	0.442	0.547
pDFR-PL2	GOV2	Titles	Oracle	0.321	0.442	0.547
pDFR-PL2	GOV2	Desc.	0	0.218	0.374	0.414
pDFR-PL2	GOV2	Desc.	1	0.266	0.421	0.522
pDFR-PL2	GOV2	Desc.	2	0.274	0.432	0.452
pDFR-PL2	GOV2	Desc.	3	0.335	0.450	0.627
pDFR-PL2	GOV2	Desc.	4	0.258	0.337	0.452
pDFR-PL2	GOV2	Desc.	Joint	0.270	0.403	0.494
pDFR-PL2	GOV2	Desc.	Avg.	0.272	0.403	0.493
pDFR-PL2	GOV2	Desc.	Oracle	0.272	0.404	0.494
pDFR-PL2	ClueWeb-09-B	Titles	0	0.134	0.294	0.455
pDFR-PL2	ClueWeb-09-B	Titles	1	0.116	0.233	0.353
pDFR-PL2	ClueWeb-09-B	Titles	2	0.124	0.314	0.400
pDFR-PL2	ClueWeb-09-B	Titles	3	0.103	0.268	0.345
pDFR-PL2	ClueWeb-09-B	Titles	4	0.077	0.132	0.238
pDFR-PL2	ClueWeb-09-B	Titles	Joint	0.111	0.248	0.358
pDFR-PL2	ClueWeb-09-B	Titles	Avg.	0.113	0.256	0.368
pDFR-PL2	ClueWeb-09-B	Titles	Oracle	0.113	0.255	0.367
pDFR-PL2	ClueWeb-09-B	Desc.	0	0.050	0.137	0.195
pDFR-PL2	ClueWeb-09-B	Desc.	1	0.086	0.201	0.237
pDFR-PL2	ClueWeb-09-B	Desc.	2	0.110	0.228	0.257
pDFR-PL2	ClueWeb-09-B	Desc.	3	0.112	0.273	0.378
pDFR-PL2	ClueWeb-09-B	Desc.	4	0.044	0.123	0.148

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
pDFR-PL2	ClueWeb-09-B	Desc.	Joint	0.080	0.193	0.244
pDFR-PL2	ClueWeb-09-B	Desc.	Avg.	0.082	0.200	0.250
pDFR-PL2	ClueWeb-09-B	Desc.	Oracle	0.082	0.200	0.250
BM25	Robust-04	Titles	0	0.261	0.396	0.351
BM25	Robust-04	Titles	1	0.261	0.401	0.343
BM25	Robust-04	Titles	2	0.263	0.461	0.411
BM25	Robust-04	Titles	3	0.253	0.412	0.359
BM25	Robust-04	Titles	4	0.230	0.390	0.351
BM25	Robust-04	Titles	Joint	0.254	0.412	0.363
BM25	Robust-04	Titles	Avg.	0.254	0.412	0.363
BM25	Robust-04	Titles	Oracle	0.255	0.413	0.362
BM25	Robust-04	Desc.	0	0.255	0.410	0.340
BM25	Robust-04	Desc.	1	0.250	0.395	0.333
BM25	Robust-04	Desc.	2	0.238	0.397	0.335
BM25	Robust-04	Desc.	3	0.219	0.389	0.353
BM25	Robust-04	Desc.	4	0.221	0.361	0.291
BM25	Robust-04	Desc.	Joint	0.237	0.390	0.331
BM25	Robust-04	Desc.	Avg.	0.238	0.394	0.334
BM25	Robust-04	Desc.	Oracle	0.239	0.391	0.331
BM25	GOV2	Titles	0	0.299	0.407	0.488
BM25	GOV2	Titles	1	0.343	0.458	0.587
BM25	GOV2	Titles	2	0.340	0.454	0.593
BM25	GOV2	Titles	3	0.256	0.405	0.455
BM25	GOV2	Titles	4	0.260	0.453	0.528
BM25	GOV2	Titles	Joint	0.299	0.435	0.530
BM25	GOV2	Titles	Avg.	0.300	0.435	0.529
BM25	GOV2	Titles	Oracle	0.301	0.436	0.531
BM25	GOV2	Desc.	0	0.206	0.362	0.414
BM25	GOV2	Desc.	1	0.272	0.428	0.507
BM25	GOV2	Desc.	2	0.252	0.433	0.447
BM25	GOV2	Desc.	3	0.323	0.427	0.595
BM25	GOV2	Desc.	4	0.248	0.352	0.453
BM25	GOV2	Desc.	Joint	0.261	0.401	0.484
BM25	GOV2	Desc.	Avg.	0.261	0.402	0.484
BM25	GOV2	Desc.	Oracle	0.261	0.400	0.484
BM25	ClueWeb-09-B	Titles	0	0.118	0.226	0.381
BM25	ClueWeb-09-B	Titles	1	0.088	0.171	0.273
BM25	ClueWeb-09-B	Titles	2	0.124	0.324	0.405
BM25	ClueWeb-09-B	Titles	3	0.089	0.243	0.300
BM25	ClueWeb-09-B	Titles	4	0.077	0.154	0.262
BM25	ClueWeb-09-B	Titles	Joint	0.099	0.223	0.324
BM25	ClueWeb-09-B	Titles	Avg.	0.101	0.230	0.334
BM25	ClueWeb-09-B	Titles	Oracle	0.102	0.226	0.329
BM25	ClueWeb-09-B	Desc.	0	0.054	0.153	0.213
BM25	ClueWeb-09-B	Desc.	1	0.079	0.206	0.258
BM25	ClueWeb-09-B	Desc.	2	0.104	0.217	0.250
BM25	ClueWeb-09-B	Desc.	3	0.115	0.280	0.380
BM25	ClueWeb-09-B	Desc.	4	0.053	0.150	0.195
BM25	ClueWeb-09-B	Desc.	Joint	0.081	0.201	0.260

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
BM25	ClueWeb-09-B	Desc.	Avg.	0.082	0.203	0.262
BM25	ClueWeb-09-B	Desc.	Oracle	0.083	0.203	0.259
BM25-TP	Robust-04	Titles	0	0.275	0.407	0.369
BM25-TP	Robust-04	Titles	1	0.273	0.426	0.370
BM25-TP	Robust-04	Titles	2	0.266	0.454	0.408
BM25-TP	Robust-04	Titles	3	0.257	0.409	0.350
BM25-TP	Robust-04	Titles	4	0.239	0.394	0.358
BM25-TP	Robust-04	Titles	Joint	0.262	0.418	0.371
BM25-TP	Robust-04	Titles	Avg.	0.263	0.417	0.368
BM25-TP	Robust-04	Titles	Oracle	0.264	0.420	0.371
BM25-TP	Robust-04	Desc.	0	0.247	0.391	0.328
BM25-TP	Robust-04	Desc.	1	0.261	0.414	0.349
BM25-TP	Robust-04	Desc.	2	0.246	0.403	0.338
BM25-TP	Robust-04	Desc.	3	0.226	0.396	0.362
BM25-TP	Robust-04	Desc.	4	0.232	0.363	0.300
BM25-TP	Robust-04	Desc.	Joint	0.243	0.394	0.336
BM25-TP	Robust-04	Desc.	Avg.	0.244	0.395	0.337
BM25-TP	Robust-04	Desc.	Oracle	0.244	0.397	0.338
BM25-TP	GOV2	Titles	0	0.328	0.439	0.533
BM25-TP	GOV2	Titles	1	0.347	0.442	0.615
BM25-TP	GOV2	Titles	2	0.343	0.463	0.612
BM25-TP	GOV2	Titles	3	0.292	0.423	0.482
BM25-TP	GOV2	Titles	4	0.296	0.460	0.540
BM25-TP	GOV2	Titles	Joint	0.321	0.445	0.556
BM25-TP	GOV2	Titles	Avg.	0.322	0.446	0.558
BM25-TP	GOV2	Titles	Oracle	0.322	0.447	0.559
BM25-TP	GOV2	Desc.	0	0.223	0.410	0.469
BM25-TP	GOV2	Desc.	1	0.294	0.411	0.508
BM25-TP	GOV2	Desc.	2	0.276	0.435	0.480
BM25-TP	GOV2	Desc.	3	0.314	0.432	0.622
BM25-TP	GOV2	Desc.	4	0.249	0.346	0.468
BM25-TP	GOV2	Desc.	Joint	0.272	0.407	0.510
BM25-TP	GOV2	Desc.	Avg.	0.273	0.409	0.510
BM25-TP	GOV2	Desc.	Oracle	0.273	0.410	0.510
BM25-TP	ClueWeb-09-B	Titles	0	0.128	0.273	0.426
BM25-TP	ClueWeb-09-B	Titles	1	0.101	0.217	0.328
BM25-TP	ClueWeb-09-B	Titles	2	0.126	0.307	0.405
BM25-TP	ClueWeb-09-B	Titles	3	0.104	0.262	0.340
BM25-TP	ClueWeb-09-B	Titles	4	0.086	0.153	0.250
BM25-TP	ClueWeb-09-B	Titles	Joint	0.109	0.242	0.349
BM25-TP	ClueWeb-09-B	Titles	Avg.	0.110	0.250	0.359
BM25-TP	ClueWeb-09-B	Titles	Oracle	0.111	0.250	0.360
BM25-TP	ClueWeb-09-B	Desc.	0	0.056	0.143	0.208
BM25-TP	ClueWeb-09-B	Desc.	1	0.085	0.223	0.250
BM25-TP	ClueWeb-09-B	Desc.	2	0.117	0.246	0.284
BM25-TP	ClueWeb-09-B	Desc.	3	0.115	0.265	0.367
BM25-TP	ClueWeb-09-B	Desc.	4	0.050	0.125	0.178
BM25-TP	ClueWeb-09-B	Desc.	Joint	0.084	0.201	0.258
BM25-TP	ClueWeb-09-B	Desc.	Avg.	0.086	0.204	0.260

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
BM25-TP	ClueWeb-09-B	Desc.	Oracle	0.086	0.204	0.260
BM25-TP2	Robust-04	Titles	0	0.266	0.390	0.339
BM25-TP2	Robust-04	Titles	1	0.266	0.425	0.359
BM25-TP2	Robust-04	Titles	2	0.241	0.415	0.370
BM25-TP2	Robust-04	Titles	3	0.244	0.377	0.332
BM25-TP2	Robust-04	Titles	4	0.223	0.374	0.338
BM25-TP2	Robust-04	Titles	Joint	0.248	0.396	0.348
BM25-TP2	Robust-04	Titles	Avg.	0.250	0.399	0.350
BM25-TP2	Robust-04	Titles	Oracle	0.250	0.399	0.350
BM25-TP2	Robust-04	Desc.	0	0.203	0.324	0.254
BM25-TP2	Robust-04	Desc.	1	0.239	0.389	0.321
BM25-TP2	Robust-04	Desc.	2	0.222	0.359	0.290
BM25-TP2	Robust-04	Desc.	3	0.203	0.358	0.335
BM25-TP2	Robust-04	Desc.	4	0.206	0.349	0.312
BM25-TP2	Robust-04	Desc.	Joint	0.215	0.356	0.302
BM25-TP2	Robust-04	Desc.	Avg.	0.216	0.358	0.303
BM25-TP2	Robust-04	Desc.	Oracle	0.216	0.359	0.305
BM25-TP2	GOV2	Titles	0	0.294	0.381	0.440
BM25-TP2	GOV2	Titles	1	0.262	0.332	0.477
BM25-TP2	GOV2	Titles	2	0.305	0.399	0.526
BM25-TP2	GOV2	Titles	3	0.261	0.395	0.478
BM25-TP2	GOV2	Titles	4	0.243	0.404	0.478
BM25-TP2	GOV2	Titles	Joint	0.273	0.382	0.480
BM25-TP2	GOV2	Titles	Avg.	0.275	0.384	0.484
BM25-TP2	GOV2	Titles	Oracle	0.275	0.387	0.488
BM25-TP2	GOV2	Desc.	0	0.187	0.349	0.378
BM25-TP2	GOV2	Desc.	1	0.244	0.403	0.488
BM25-TP2	GOV2	Desc.	2	0.233	0.381	0.412
BM25-TP2	GOV2	Desc.	3	0.285	0.417	0.597
BM25-TP2	GOV2	Desc.	4	0.183	0.249	0.357
BM25-TP2	GOV2	Desc.	Joint	0.227	0.360	0.447
BM25-TP2	GOV2	Desc.	Avg.	0.229	0.358	0.448
BM25-TP2	GOV2	Desc.	Oracle	0.229	0.361	0.451
BM25-TP2	ClueWeb-09-B	Titles	0	0.114	0.266	0.409
BM25-TP2	ClueWeb-09-B	Titles	1	0.092	0.219	0.305
BM25-TP2	ClueWeb-09-B	Titles	2	0.124	0.304	0.408
BM25-TP2	ClueWeb-09-B	Titles	3	0.097	0.254	0.328
BM25-TP2	ClueWeb-09-B	Titles	4	0.076	0.145	0.210
BM25-TP2	ClueWeb-09-B	Titles	Joint	0.100	0.237	0.331
BM25-TP2	ClueWeb-09-B	Titles	Avg.	0.102	0.239	0.333
BM25-TP2	ClueWeb-09-B	Titles	Oracle	0.102	0.239	0.331
BM25-TP2	ClueWeb-09-B	Desc.	0	0.050	0.120	0.168
BM25-TP2	ClueWeb-09-B	Desc.	1	0.071	0.198	0.223
BM25-TP2	ClueWeb-09-B	Desc.	2	0.084	0.206	0.250
BM25-TP2	ClueWeb-09-B	Desc.	3	0.108	0.267	0.357
BM25-TP2	ClueWeb-09-B	Desc.	4	0.049	0.115	0.157
BM25-TP2	ClueWeb-09-B	Desc.	Joint	0.073	0.182	0.231
BM25-TP2	ClueWeb-09-B	Desc.	Avg.	0.074	0.185	0.236
BM25-TP2	ClueWeb-09-B	Desc.	Oracle	0.075	0.186	0.239

Continued on next page

Continued from previous page

Model	Coll.	Query Set	Fold	MAP	nDCG@20	P@20
BM25-Span	Robust-04	Titles	0	0.271	0.402	0.357
BM25-Span	Robust-04	Titles	1	0.271	0.419	0.361
BM25-Span	Robust-04	Titles	2	0.272	0.461	0.410
BM25-Span	Robust-04	Titles	3	0.262	0.425	0.369
BM25-Span	Robust-04	Titles	4	0.238	0.405	0.370
BM25-Span	Robust-04	Titles	Joint	0.263	0.423	0.373
BM25-Span	Robust-04	Titles	Avg.	0.264	0.423	0.373
BM25-Span	Robust-04	Titles	Oracle	0.264	0.424	0.374
BM25-Span	Robust-04	Desc.	0	0.268	0.416	0.348
BM25-Span	Robust-04	Desc.	1	0.251	0.398	0.332
BM25-Span	Robust-04	Desc.	2	0.243	0.404	0.332
BM25-Span	Robust-04	Desc.	3	0.224	0.397	0.367
BM25-Span	Robust-04	Desc.	4	0.229	0.357	0.285
BM25-Span	Robust-04	Desc.	Joint	0.243	0.394	0.333
BM25-Span	Robust-04	Desc.	Avg.	0.244	0.395	0.333
BM25-Span	Robust-04	Desc.	Oracle	0.244	0.394	0.333
BM25-Span	GOV2	Titles	0	0.337	0.429	0.527
BM25-Span	GOV2	Titles	1	0.372	0.461	0.603
BM25-Span	GOV2	Titles	2	0.369	0.462	0.602
BM25-Span	GOV2	Titles	3	0.297	0.427	0.508
BM25-Span	GOV2	Titles	4	0.303	0.478	0.560
BM25-Span	GOV2	Titles	Joint	0.336	0.451	0.560
BM25-Span	GOV2	Titles	Avg.	0.336	0.450	0.558
BM25-Span	GOV2	Titles	Oracle	0.336	0.452	0.561
BM25-Span	GOV2	Desc.	0	0.220	0.381	0.445
BM25-Span	GOV2	Desc.	1	0.287	0.438	0.540
BM25-Span	GOV2	Desc.	2	0.271	0.438	0.480
BM25-Span	GOV2	Desc.	3	0.335	0.445	0.623
BM25-Span	GOV2	Desc.	4	0.254	0.349	0.462
BM25-Span	GOV2	Desc.	Joint	0.274	0.410	0.510
BM25-Span	GOV2	Desc.	Avg.	0.275	0.413	0.514
BM25-Span	GOV2	Desc.	Oracle	0.275	0.413	0.514
BM25-Span	ClueWeb-09-B	Titles	0	0.120	0.216	0.372
BM25-Span	ClueWeb-09-B	Titles	1	0.101	0.208	0.332
BM25-Span	ClueWeb-09-B	Titles	2	0.125	0.328	0.432
BM25-Span	ClueWeb-09-B	Titles	3	0.099	0.246	0.317
BM25-Span	ClueWeb-09-B	Titles	4	0.082	0.134	0.225
BM25-Span	ClueWeb-09-B	Titles	Joint	0.105	0.226	0.335
BM25-Span	ClueWeb-09-B	Titles	Avg.	0.110	0.248	0.356
BM25-Span	ClueWeb-09-B	Titles	Oracle	0.110	0.249	0.360
BM25-Span	ClueWeb-09-B	Desc.	0	0.055	0.160	0.215
BM25-Span	ClueWeb-09-B	Desc.	1	0.084	0.202	0.240
BM25-Span	ClueWeb-09-B	Desc.	2	0.117	0.244	0.279
BM25-Span	ClueWeb-09-B	Desc.	3	0.115	0.277	0.387
BM25-Span	ClueWeb-09-B	Desc.	4	0.054	0.140	0.181
BM25-Span	ClueWeb-09-B	Desc.	Joint	0.085	0.205	0.261
BM25-Span	ClueWeb-09-B	Desc.	Avg.	0.087	0.212	0.272
BM25-Span	ClueWeb-09-B	Desc.	Oracle	0.088	0.216	0.272

Statistical Tests

In this final section, we present p-values computed using Fisher's randomization test to compare the performance of QL, SDM, and WSDM-Internal, to each of the other retrieval models. The statistical test is performed over the joint results, for each collection and query set, for the MAP metric. The collections, query sets and this retrieval metric are all defined in Chapter 3. Significant improvements over the paired model, for each comparison pair, is highlighted in bold.

Model 1	Model 2	Coll.	Query Set	p-value
Robust-04, Titles				
QL	**SDM**	Robust-04	Titles	0.000
QL	**Uni+O234**	Robust-04	Titles	0.001
QL	**Uni+O234+U2**	Robust-04	Titles	0.000
QL	**Uni+O23+U23**	Robust-04	Titles	0.000
QL	**WSDM**	Robust-04	Titles	0.000
QL	**WSDM-Int**	Robust-04	Titles	0.000
QL	**WSDM-Int-3**	Robust-04	Titles	0.000
QL	PLM	Robust-04	Titles	0.487
QL	**PLM-2**	Robust-04	Titles	0.001
QL	PL2	Robust-04	Titles	0.232
QL	**pDFR-BiL2**	Robust-04	Titles	0.006
QL	**pDFR-PL2**	Robust-04	Titles	0.003
QL	BM25	Robust-04	Titles	0.216
QL	**BM25-TP**	Robust-04	Titles	0.005
QL	BM25-TP2	Robust-04	Titles	0.775
QL	**BM25-Span**	Robust-04	Titles	0.000
SDM	Uni+O234	Robust-04	Titles	0.911
SDM	Uni+O234+U2	Robust-04	Titles	0.570
SDM	Uni+O23+U23	Robust-04	Titles	0.375
SDM	**WSDM**	Robust-04	Titles	0.000
SDM	**WSDM-Int**	Robust-04	Titles	0.003
SDM	**WSDM-Int-3**	Robust-04	Titles	0.000
SDM	PLM	Robust-04	Titles	1.000
SDM	PLM-2	Robust-04	Titles	0.991
SDM	PL2	Robust-04	Titles	0.998
SDM	pDFR-BiL2	Robust-04	Titles	0.963
SDM	pDFR-PL2	Robust-04	Titles	0.933
SDM	BM25	Robust-04	Titles	0.991
SDM	BM25-TP	Robust-04	Titles	0.628
SDM	BM25-TP2	Robust-04	Titles	1.000
SDM	BM25-Span	Robust-04	Titles	0.520
WSDM-Int	Uni+O234	Robust-04	Titles	1.000
WSDM-Int	Uni+O234+U2	Robust-04	Titles	0.998
WSDM-Int	Uni+O23+U23	Robust-04	Titles	0.998
WSDM-Int	**WSDM**	Robust-04	Titles	0.000
WSDM-Int	**WSDM-Int-3**	Robust-04	Titles	0.001

Continued on next page

Continued from previous page

Model (Baseline)	Model (Treatment)	Coll.	Query Set	p-value
WSDM-Int	PLM	Robust-04	Titles	1.000
WSDM-Int	PLM-2	Robust-04	Titles	1.000
WSDM-Int	PL2	Robust-04	Titles	1.000
WSDM-Int	pDFR-BiL2	Robust-04	Titles	0.999
WSDM-Int	pDFR-PL2	Robust-04	Titles	0.999
WSDM-Int	BM25	Robust-04	Titles	1.000
WSDM-Int	BM25-TP	Robust-04	Titles	0.985
WSDM-Int	BM25-TP2	Robust-04	Titles	1.000
WSDM-Int	BM25-Span	Robust-04	Titles	0.959
Robust-04, Desc.				
QL	**SDM**	Robust-04	Desc.	0.000
QL	**Uni+O234**	Robust-04	Desc.	0.000
QL	**Uni+O234+U2**	Robust-04	Desc.	0.000
QL	**Uni+O23+U23**	Robust-04	Desc.	0.000
QL	**Uni+O234+U234**	Robust-04	Desc.	0.000
QL	**WSDM**	Robust-04	Desc.	0.000
QL	**WSDM-Int**	Robust-04	Desc.	0.000
QL	**WSDM-Int-3**	Robust-04	Desc.	0.000
QL	PLM	Robust-04	Desc.	0.051
QL	**PLM-2**	Robust-04	Desc.	0.000
QL	PL2	Robust-04	Desc.	1.000
QL	pDFR-BiL2	Robust-04	Desc.	0.974
QL	pDFR-PL2	Robust-04	Desc.	0.951
QL	BM25	Robust-04	Desc.	0.962
QL	BM25-TP	Robust-04	Desc.	0.601
QL	BM25-TP2	Robust-04	Desc.	1.000
QL	BM25-Span	Robust-04	Desc.	0.587
SDM	Uni+O234	Robust-04	Desc.	0.559
SDM	**Uni+O234+U2**	Robust-04	Desc.	0.026
SDM	Uni+O23+U23	Robust-04	Desc.	0.100
SDM	Uni+O234+U234	Robust-04	Desc.	0.151
SDM	**WSDM**	Robust-04	Desc.	0.000
SDM	**WSDM-Int**	Robust-04	Desc.	0.000
SDM	**WSDM-Int-3**	Robust-04	Desc.	0.000
SDM	PLM	Robust-04	Desc.	0.982
SDM	PLM-2	Robust-04	Desc.	0.289
SDM	PL2	Robust-04	Desc.	1.000
SDM	pDFR-BiL2	Robust-04	Desc.	1.000
SDM	pDFR-PL2	Robust-04	Desc.	1.000
SDM	BM25	Robust-04	Desc.	1.000
SDM	BM25-TP	Robust-04	Desc.	0.999
SDM	BM25-TP2	Robust-04	Desc.	1.000
SDM	BM25-Span	Robust-04	Desc.	1.000
WSDM-Int	Uni+O234	Robust-04	Desc.	1.000
WSDM-Int	Uni+O234+U2	Robust-04	Desc.	1.000
WSDM-Int	Uni+O23+U23	Robust-04	Desc.	1.000
WSDM-Int	Uni+O234+U234	Robust-04	Desc.	1.000
WSDM-Int	**WSDM**	Robust-04	Desc.	0.000
WSDM-Int	**WSDM-Int-3**	Robust-04	Desc.	0.049

Continued on next page

Continued from previous page

Model (Baseline)	Model (Treatment)	Coll.	Query Set	p-value
WSDM-Int	PLM	Robust-04	Desc.	1.000
WSDM-Int	PLM-2	Robust-04	Desc.	1.000
WSDM-Int	PL2	Robust-04	Desc.	1.000
WSDM-Int	pDFR-BiL2	Robust-04	Desc.	1.000
WSDM-Int	pDFR-PL2	Robust-04	Desc.	1.000
WSDM-Int	BM25	Robust-04	Desc.	1.000
WSDM-Int	BM25-TP	Robust-04	Desc.	1.000
WSDM-Int	BM25-TP2	Robust-04	Desc.	1.000
WSDM-Int	BM25-Span	Robust-04	Desc.	1.000
	GOV2, Titles			
QL	**SDM**	GOV2	Titles	0.000
QL	**Uni+O234**	GOV2	Titles	0.000
QL	**Uni+O234+U2**	GOV2	Titles	0.000
QL	**Uni+O23+U23**	GOV2	Titles	0.000
QL	**WSDM**	GOV2	Titles	0.000
QL	**WSDM-Int**	GOV2	Titles	0.000
QL	**WSDM-Int-3**	GOV2	Titles	0.000
QL	**PLM**	GOV2	Titles	0.007
QL	PLM-2	GOV2	Titles	0.171
QL	PL2	GOV2	Titles	0.301
QL	**pDFR-BiL2**	GOV2	Titles	0.000
QL	**pDFR-PL2**	GOV2	Titles	0.000
QL	BM25	GOV2	Titles	0.393
QL	**BM25-TP**	GOV2	Titles	0.003
QL	BM25-TP2	GOV2	Titles	0.974
QL	**BM25-Span**	GOV2	Titles	0.000
SDM	Uni+O234	GOV2	Titles	1.000
SDM	Uni+O234+U2	GOV2	Titles	0.972
SDM	Uni+O23+U23	GOV2	Titles	0.662
SDM	**WSDM**	GOV2	Titles	0.038
SDM	WSDM-Int	GOV2	Titles	0.160
SDM	WSDM-Int-3	GOV2	Titles	0.091
SDM	PLM	GOV2	Titles	1.000
SDM	PLM-2	GOV2	Titles	1.000
SDM	PL2	GOV2	Titles	1.000
SDM	pDFR-BiL2	GOV2	Titles	0.996
SDM	pDFR-PL2	GOV2	Titles	0.994
SDM	BM25	GOV2	Titles	1.000
SDM	BM25-TP	GOV2	Titles	0.785
SDM	BM25-TP2	GOV2	Titles	1.000
SDM	**BM25-Span**	GOV2	Titles	0.024
WSDM-Int	Uni+O234	GOV2	Titles	1.000
WSDM-Int	Uni+O234+U2	GOV2	Titles	0.929
WSDM-Int	Uni+O23+U23	GOV2	Titles	0.847
WSDM-Int	**WSDM**	GOV2	Titles	0.021
WSDM-Int	WSDM-Int-3	GOV2	Titles	0.237
WSDM-Int	PLM	GOV2	Titles	1.000
WSDM-Int	PLM-2	GOV2	Titles	1.000
WSDM-Int	PL2	GOV2	Titles	1.000

Continued on next page

Continued from previous page

Model (Baseline)	Model (Treatment)	Coll.	Query Set	p-value
WSDM-Int	pDFR-BiL2	GOV2	Titles	0.999
WSDM-Int	pDFR-PL2	GOV2	Titles	0.997
WSDM-Int	BM25	GOV2	Titles	1.000
WSDM-Int	BM25-TP	GOV2	Titles	0.936
WSDM-Int	BM25-TP2	GOV2	Titles	1.000
WSDM-Int	BM25-Span	GOV2	Titles	0.089
GOV2, Desc.				
QL	**SDM**	GOV2	Desc.	0.000
QL	**Uni+O234**	GOV2	Desc.	0.000
QL	**Uni+O234+U2**	GOV2	Desc.	0.000
QL	**Uni+O23+U23**	GOV2	Desc.	0.000
QL	**Uni+O234+U234**	GOV2	Desc.	0.000
QL	**WSDM**	GOV2	Desc.	0.000
QL	**WSDM-Int**	GOV2	Desc.	0.000
QL	**WSDM-Int-3**	GOV2	Desc.	0.000
QL	PLM	GOV2	Desc.	0.945
QL	**PLM-2**	GOV2	Desc.	0.000
QL	PL2	GOV2	Desc.	0.352
QL	**pDFR-BiL2**	GOV2	Desc.	0.020
QL	**pDFR-PL2**	GOV2	Desc.	0.002
QL	BM25	GOV2	Desc.	0.267
QL	**BM25-TP**	GOV2	Desc.	0.017
QL	BM25-TP2	GOV2	Desc.	0.998
QL	**BM25-Span**	GOV2	Desc.	0.003
SDM	Uni+O234	GOV2	Desc.	0.996
SDM	Uni+O234+U2	GOV2	Desc.	0.358
SDM	Uni+O23+U23	GOV2	Desc.	0.896
SDM	Uni+O234+U234	GOV2	Desc.	0.809
SDM	**WSDM**	GOV2	Desc.	0.000
SDM	**WSDM-Int**	GOV2	Desc.	0.000
SDM	**WSDM-Int-3**	GOV2	Desc.	0.000
SDM	PLM	GOV2	Desc.	1.000
SDM	PLM-2	GOV2	Desc.	0.947
SDM	PL2	GOV2	Desc.	1.000
SDM	pDFR-BiL2	GOV2	Desc.	1.000
SDM	pDFR-PL2	GOV2	Desc.	0.999
SDM	BM25	GOV2	Desc.	0.999
SDM	BM25-TP	GOV2	Desc.	0.988
SDM	BM25-TP2	GOV2	Desc.	1.000
SDM	BM25-Span	GOV2	Desc.	0.948
WSDM-Int	Uni+O234	GOV2	Desc.	1.000
WSDM-Int	Uni+O234+U2	GOV2	Desc.	1.000
WSDM-Int	Uni+O23+U23	GOV2	Desc.	1.000
WSDM-Int	Uni+O234+U234	GOV2	Desc.	1.000
WSDM-Int	**WSDM**	GOV2	Desc.	0.029
WSDM-Int	WSDM-Int-3	GOV2	Desc.	0.860
WSDM-Int	PLM	GOV2	Desc.	1.000
WSDM-Int	PLM-2	GOV2	Desc.	1.000
WSDM-Int	PL2	GOV2	Desc.	1.000

Continued on next page

Continued from previous page

Model (Baseline)	Model (Treatment)	Coll.	Query Set	p-value
WSDM-Int	pDFR-BiL2	GOV2	Desc.	1.000
WSDM-Int	pDFR-PL2	GOV2	Desc.	1.000
WSDM-Int	BM25	GOV2	Desc.	1.000
WSDM-Int	BM25-TP	GOV2	Desc.	1.000
WSDM-Int	BM25-TP2	GOV2	Desc.	1.000
WSDM-Int	BM25-Span	GOV2	Desc.	1.000
ClueWeb-09-B, Titles				
QL	**SDM**	ClueWeb-09-B	Titles	0.000
QL	**Uni+O234**	ClueWeb-09-B	Titles	0.000
QL	**Uni+O234+U2**	ClueWeb-09-B	Titles	0.000
QL	**Uni+O23+U23**	ClueWeb-09-B	Titles	0.002
QL	**WSDM**	ClueWeb-09-B	Titles	0.000
QL	**WSDM-Int**	ClueWeb-09-B	Titles	0.000
QL	**WSDM-Int-3**	ClueWeb-09-B	Titles	0.000
QL	PLM	ClueWeb-09-B	Titles	0.944
QL	PLM-2	ClueWeb-09-B	Titles	0.604
QL	**PL2**	ClueWeb-09-B	Titles	0.000
QL	**pDFR-BiL2**	ClueWeb-09-B	Titles	0.000
QL	**pDFR-PL2**	ClueWeb-09-B	Titles	0.000
QL	BM25	ClueWeb-09-B	Titles	0.365
QL	**BM25-TP**	ClueWeb-09-B	Titles	0.002
QL	BM25-TP2	ClueWeb-09-B	Titles	0.301
QL	**BM25-Span**	ClueWeb-09-B	Titles	0.025
SDM	Uni+O234	ClueWeb-09-B	Titles	0.642
SDM	Uni+O234+U2	ClueWeb-09-B	Titles	0.229
SDM	Uni+O23+U23	ClueWeb-09-B	Titles	0.595
SDM	**WSDM**	ClueWeb-09-B	Titles	0.030
SDM	**WSDM-Int**	ClueWeb-09-B	Titles	0.000
SDM	WSDM-Int-3	ClueWeb-09-B	Titles	0.067
SDM	PLM	ClueWeb-09-B	Titles	1.000
SDM	PLM-2	ClueWeb-09-B	Titles	1.000
SDM	PL2	ClueWeb-09-B	Titles	0.842
SDM	pDFR-BiL2	ClueWeb-09-B	Titles	0.984
SDM	**pDFR-PL2**	ClueWeb-09-B	Titles	0.037
SDM	BM25	ClueWeb-09-B	Titles	0.979
SDM	BM25-TP	ClueWeb-09-B	Titles	0.391
SDM	BM25-TP2	ClueWeb-09-B	Titles	0.962
SDM	BM25-Span	ClueWeb-09-B	Titles	0.774
WSDM-Int	Uni+O234	ClueWeb-09-B	Titles	1.000
WSDM-Int	Uni+O234+U2	ClueWeb-09-B	Titles	0.973
WSDM-Int	Uni+O23+U23	ClueWeb-09-B	Titles	0.990
WSDM-Int	WSDM	ClueWeb-09-B	Titles	0.985
WSDM-Int	WSDM-Int-3	ClueWeb-09-B	Titles	0.891
WSDM-Int	PLM	ClueWeb-09-B	Titles	1.000
WSDM-Int	PLM-2	ClueWeb-09-B	Titles	1.000
WSDM-Int	PL2	ClueWeb-09-B	Titles	0.999
WSDM-Int	pDFR-BiL2	ClueWeb-09-B	Titles	1.000
WSDM-Int	pDFR-PL2	ClueWeb-09-B	Titles	0.739
WSDM-Int	BM25	ClueWeb-09-B	Titles	0.999

Continued on next page

Continued from previous page

Model (Baseline)	Model (Treatment)	Coll.	Query Set	p-value
WSDM-Int	BM25-TP	ClueWeb-09-B	Titles	0.852
WSDM-Int	BM25-TP2	ClueWeb-09-B	Titles	0.998
WSDM-Int	BM25-Span	ClueWeb-09-B	Titles	0.973
ClueWeb-09-B, Desc.				
QL	SDM	ClueWeb-09-B	Desc.	0.250
QL	Uni+O234	ClueWeb-09-B	Desc.	0.498
QL	Uni+O234+U2	ClueWeb-09-B	Desc.	0.055
QL	Uni+O23+U23	ClueWeb-09-B	Desc.	0.402
QL	Uni+O234+U234	ClueWeb-09-B	Desc.	0.118
QL	**WSDM**	ClueWeb-09-B	Desc.	0.000
QL	**WSDM-Int**	ClueWeb-09-B	Desc.	0.003
QL	**WSDM-Int-3**	ClueWeb-09-B	Desc.	0.026
QL	PLM	ClueWeb-09-B	Desc.	0.999
QL	PLM-2	ClueWeb-09-B	Desc.	0.315
QL	**PL2**	ClueWeb-09-B	Desc.	0.025
QL	pDFR-BiL2	ClueWeb-09-B	Desc.	0.107
QL	**pDFR-PL2**	ClueWeb-09-B	Desc.	0.015
QL	**BM25**	ClueWeb-09-B	Desc.	0.012
QL	**BM25-TP**	ClueWeb-09-B	Desc.	0.000
QL	BM25-TP2	ClueWeb-09-B	Desc.	0.583
QL	**BM25-Span**	ClueWeb-09-B	Desc.	0.000
SDM	Uni+O234	ClueWeb-09-B	Desc.	0.991
SDM	Uni+O234+U2	ClueWeb-09-B	Desc.	0.238
SDM	Uni+O23+U23	ClueWeb-09-B	Desc.	0.931
SDM	Uni+O234+U234	ClueWeb-09-B	Desc.	0.246
SDM	**WSDM**	ClueWeb-09-B	Desc.	0.003
SDM	WSDM-Int	ClueWeb-09-B	Desc.	0.147
SDM	WSDM-Int-3	ClueWeb-09-B	Desc.	0.290
SDM	PLM	ClueWeb-09-B	Desc.	0.999
SDM	PLM-2	ClueWeb-09-B	Desc.	0.641
SDM	PL2	ClueWeb-09-B	Desc.	0.549
SDM	pDFR-BiL2	ClueWeb-09-B	Desc.	0.607
SDM	pDFR-PL2	ClueWeb-09-B	Desc.	0.313
SDM	BM25	ClueWeb-09-B	Desc.	0.271
SDM	BM25-TP	ClueWeb-09-B	Desc.	0.071
SDM	BM25-TP2	ClueWeb-09-B	Desc.	0.901
SDM	**BM25-Span**	ClueWeb-09-B	Desc.	0.025
WSDM-Int	Uni+O234	ClueWeb-09-B	Desc.	0.977
WSDM-Int	Uni+O234+U2	ClueWeb-09-B	Desc.	0.816
WSDM-Int	Uni+O23+U23	ClueWeb-09-B	Desc.	0.936
WSDM-Int	Uni+O234+U234	ClueWeb-09-B	Desc.	0.782
WSDM-Int	**WSDM**	ClueWeb-09-B	Desc.	0.005
WSDM-Int	WSDM-Int-3	ClueWeb-09-B	Desc.	0.958
WSDM-Int	PLM	ClueWeb-09-B	Desc.	1.000
WSDM-Int	PLM-2	ClueWeb-09-B	Desc.	0.996
WSDM-Int	PL2	ClueWeb-09-B	Desc.	0.957
WSDM-Int	pDFR-BiL2	ClueWeb-09-B	Desc.	0.980
WSDM-Int	pDFR-PL2	ClueWeb-09-B	Desc.	0.718
WSDM-Int	BM25	ClueWeb-09-B	Desc.	0.645

Continued on next page

Continued from previous page

Model (Baseline)	Model (Treatment)	Coll.	Query Set	p-value
WSDM-Int	BM25-TP	ClueWeb-09-B	Desc.	0.289
WSDM-Int	BM25-TP2	ClueWeb-09-B	Desc.	0.969
WSDM-Int	BM25-Span	ClueWeb-09-B	Desc.	0.189

BIBLIOGRAPHY

Elif Aktolga, James Allan, and David A. Smith. Passage reranking for question answering using syntactic structures and answer types. In *Proc. of the 33rd ECIR*, pages 617–628, 2011.

N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.

Gianni Amati and Cornelis Joost Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Trans. Inf. Syst.*, 20(4):357–389, October 2002.

Vo Ngoc Anh and Alistair Moffat. Improved retrieval effectiveness through impact transformation. *Aust. Comput. Sci. Commun.*, 24(2):41–47, January 2002a.

Vo Ngoc Anh and Alistair Moffat. Impact transformation: effective and efficient web retrieval. In *Proc. of the 25th ACM SIGIR*, pages 3–10, 2002b.

Vo Ngoc Anh and Alistair Moffat. Inverted index compression using word-aligned binary codes. *Inf. Retr.*, 8(1):151–166, January 2005. ISSN 1386-4564.

Vo Ngoc Anh and Alistair Moffat. Pruned query evaluation using pre-computed impacts. In *Proceedings of the 29th ACM SIGIR*, pages 372–379, 2006.

T. G. Armstrong, A. Moffat, W. Webber, and J. Zobel. Improvements that don't add up: Ad-hoc retrieval results since 1998. In *Proc. 18th ACM CIKM*, pages 601–610, November 2009a.

Timothy G. Armstrong, Alistair Moffat, William Webber, and Justin Zobel. Has adhoc retrieval improved since 1994? In *Proc. of the 32nd ACM SIGIR*, pages 692–693, 2009b.

Diego Arroyuelo, Senén González, Mauricio Oyarzún, and Victor Sepulveda. Document identifier reassignment and run-length-compressed inverted indexes for improved search performance. In *Proc. of the 36th ACM SIGIR*, pages 173–182, 2013.

Ricardo Baeza-Yates and Felipe Saint-Jean. A three level search engine index based in query log distribution. In *String Processing and Information Retrieval*, volume 2857, pages 56–65. Springer Berlin / Heidelberg, 2003.

Ricardo Baeza-Yates, Flavio Junqueira, Vassilis Plachouras, and Hans Friedrich Witschel. Admission policies for caches of search engine results. In *Proceedings of the 14th SPIRE*, pages 74–85, 2007.

Dirk Bahle, Hugh E. Williams, and Justin Zobel. Efficient phrase querying with an auxiliary index. In *SIGIR*, pages 215–221, 2002.

Jing Bai, Yi Chang, Hang Cui, Zhaohui Zheng, Gordon Sun, and Xin Li. Investigation of partial query proximity in web search. In *Proc. of the 17th WWW*, pages 1183–1184, 2008.

M. Bendersky and W.B. Croft. Discovering key concepts in verbose queries. In *Proceedings of the 31st ACM SIGIR*, pages 491–498. ACM, 2008.

Michael Bendersky and W. Bruce Croft. Analysis of long queries in a large scale search log. In *Proceedings of the 2009 Workshop on Web Search Click Data*, WSCD '09, pages 8–14, 2009.

Michael Bendersky, Donald Metzler, and W. Bruce Croft. Learning concept importance using a weighted dependence model. In *Proceedings of the third ACM WSDM*, pages 31–40, 2010.

Michael Bendersky, Donald Metzler, and W. Bruce Croft. Parameterized concept weighting in verbose queries. In *Proc. of the 34th ACM SIGIR*, pages 605–614, 2011.

Michael Bendersky, Donald Metzler, and W. Bruce Croft. Effective query formulation with multiple information sources. In *Proc. of the fifth ACM WSDM*, pages 443–452, 2012.

S. Bergsma and Q.I. Wang. Learning noun phrase query segmentation. In *Proc. of the EMNLP-CoNLL*, pages 819–826, 2007.

R. Berinde, G. Cormode, P. Indyk, and M. J. Strauss. Space-optimal heavy hitters with strong error bounds. In *PODS*, pages 157–166, 2009.

Y. Bernstein and J. Zobel. Accurate discovery of co-derivative documents via duplicate text detection. *Information Systems*, 31:595–609, 2006.

Sergey Brin, James Davis, and Héctor García-Molina. Copy detection mechanisms for digital documents. In *Proc. of the 1995 ACM SIGMOD*, pages 398–409, 1995.

Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *Proc. of the 12th CIKM*, pages 426–434, 2003.

Andreas Broschart and Ralf Schenkel. High-performance processing of text queries with tunable pruned term and term pair indexes. *ACM Trans. Inf. Syst.*, 30(1): 5:1–5:32, March 2012.

Chris Buckley and Alan F. Lewit. Optimization of inverted vector searches. In *Proc. of the 8th ACM SIGIR*, SIGIR '85, pages 97–110, 1985.

M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Equipment Corporation, 1994.

S. Büttcher, C. Clarke, and G.V. Cormack. *Information Retrieval: Implementing and evaluating search engines*. The MIT Press, 2010.

Stefan Büttcher, Charles L. A. Clarke, and Brad Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. In *Proceedings of the 29th ACM SIGIR*, pages 621–622, 2006.

James P. Callan, W. Bruce Croft, and John Broglio. TREC and TIPSTER Experiments with INQUERY. In *Information Processing & Management*, pages 31–3. Morgan Kaufmann, 1994.

Berkant Barla Cambazoglu, Flavio P. Junqueira, Vassilis Plachouras, Scott Banachowski, Baoqiu Cui, Swee Lim, and Bill Bridge. A refreshing perspective of search engine caching. In *Proceedings of the 19th WWW*, pages 181–190, 2010.

M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Automata, Languages and Programming*, pages 784–784, 2002.

William S. Cooper. Some inconsistencies and misnomers in probabilistic information retrieval. In *Proc. of the 14th ACM SIGIR*, pages 57–61, 1991.

Gordon V. Cormack, Mark D. Smucker, and Charles L. A. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *CoRR*, abs/1004.5168, 2010.

G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. *Publication of the VLDB Endowment*, 1(2):1530–1541, 2008.

G. Cormode and M. Hadjieleftheriou. Methods for finding frequent items in data streams. *The VLDB Journal*, 19(1):3–20, 2010.

G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *Proc. 6th Latin American Symposium on Theoretical Informatics*, pages 29–38, 2004.

G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005a.

G. Cormode and S. Muthukrishnan. Summarizing and mining skewed data streams. In *Proc. of the 5th SIAM Conference on Data Mining*, 2005b.

W.B. Croft, D. Metzler, and T. Strohman. *Search engines: Information retrieval in practice*. Addison-Wesley, 2010.

J. S. Culpepper, M. Yasukawa, and F. Scholer. Language independent ranked retrieval with NeWT. In *ADCS*, pages 18–25, December 2011.

J. S. Culpepper, M. Petri, and F. Scholer. Efficient in-memory top-k document retrieval. In *SIGIR*, pages 225–234, 2012.

J. Shane Culpepper and Alistair Moffat. Efficient set intersection for inverted indexing. *ACM Trans. Inf. Syst.*, 29(1):1:1–1:25, December 2010.

Peter Elias. Universal codeword sets and representations of the integers. *IEEE Trans. on Information Theory*, 21:194–203, March 1975.

C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *SIGCOMM Comput. Commun. Rev.*, pages 323–336, 2002.

Tiziano Fagni, Raffaele Perego, Fabrizio Silvestri, and Salvatore Orlando. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24(1):51–78, January 2006.

C. Faloutsos. Signature files. In *Information Retrieval: Data Structures & Algorithms*, pages 44–65. Prentice Hall, New Jersey, 1992.

Antonio Fariña, Nieves R. Brisaboa, Gonzalo Navarro, Francisco Claude, Ángeles S. Places, and Eduardo Rodríguez. Word-based self-indexes for natural language text. *ACM Trans. Inf. Syst.*, 30(1):1:1–1:34, March 2012.

P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proc. of the 41st FOCS*, pages 390–, 2000.

M. Fontoura, V. Josifovski, J. Liu, S. Venkatesan, X. Zhu, and J. Zien. Evaluation strategies for top-k queries over memory-resident inverted indexes. *Proceedings of the VLDB Endowment*, 4(12):1213–1224, 2011.

Qingqing Gan and Torsten Suel. Improved techniques for result caching in web search engines. In *Proceedings of the 18th WWW*, pages 431–440, 2009.

S. Ganguly, M. N. Garofalakis, and R. Rastogi. Processing data-stream join aggregates using skimmed sketches. In *In Proc. Int. Conf. on Extending Database Technology (EDBT)*, pages 569–586, 2004.

Jianfeng Gao, Jian-Yun Nie, Guangyuan Wu, and Guihong Cao. Dependence language model for information retrieval. In *Proceedings of the 27th ACM SIGIR*, pages 170–177, 2004.

S.W. Golomb. Run-length encodings. *IEEE Trans. on Information Theory*, 12: 399–401, 1966.

Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract). In *Proc. of the 32nd ACM STOC*, STOC '00, pages 397–406, 2000.

Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proc. of the 14th SODA*, pages 841–850, 2003. ISBN 0-89871-538-5.

Paul Haahr, Laramie Leavitt, John Sarapata, Trevor Strohman, and Robert M. Wyman. Predictive searching and associated cache management, 06 2009. Patent Application US20100318538 A1.

W.-K. Hon, R. Shah, and J. S. Vitter. Space-efficient framework for top-k string retrieval problems. In *FOCS*, pages 713–722, 2009.

W.-K. Hon, R. Shah, and S. V. Thankachan. Towards an optimal space-and-query-time index for top-k document retrieval. In *CPM*, pages 673–184, 2012.

S. Huston, A. Moffat, and W.B. Croft. Efficient indexing of repeated n-grams. In *Proc. of the 4th ACM WSDM*, pages 127–136. ACM, 2011.

S. Huston, J. Shane Culpepper, and W.B. Croft. Sketch-based indexing of term dependency statistics. In *Proc. of the 21st ACM CKIM*, 2012.

S. Huston, J. Shane Culpepper, and W.B. Croft. Indexing word-sequences for ranked retrieval. In *TOIS*, 2014.

Samuel Huston and W. Bruce Croft. Evaluating verbose query processing techniques. In *Proc. of the 33rd ACM SIGIR*, pages 291–298, 2010.

Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *Proc. of the 15th WWW*, pages 387–396, 2006.

Alan J. Kent, Ron Sacks-Davis, and Kotagiri Ramamohanarao. A signature file scheme based on multiple organizations for indexing very large text databases. *JASIS*, 41(7):508–534, 1990.

Robert Krovetz. Viewing morphology as an inference process. In *Proc. of the 16th ACM SIGIR*, pages 191–202, 1993.

Matthew Lease. An improved markov random field model for supporting verbose queries. In *Proc. of the 32nd ACM SIGIR*, pages 476–483, 2009.

Xiaohui Long and Torsten Suel. Three-level caching for efficient query processing in large web search engines. In *Proceedings of the 14th WWW*, pages 257–266, 2005.

Robert M. Losee. Term dependence: Truncating the bahadur lazarsfeld expansion. *Information Processing and Management*, 30(2):293 – 303, 1994.

Y. Lv and C.X. Zhai. Positional language models for information retrieval. In *Proc. of the 32nd ACM SIGIR*, pages 299–306. ACM, 2009.

C. Macdonald, B. He, V. Plachouras, and I. Ounis. University of glasgow at trec 2005: Experiments in terabyte and enterprise tracks with terrier. In *Proceedings of TREC 2005*, 2005.

Craig Macdonald, Iadh Ounis, and Nicola Tonellotto. Upper-bound approximations for dynamic pruning. *ACM Trans. Inf. Syst.*, 29(4):17, 2011a.

Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. On upper bounds for dynamic pruning. In *ICTIR*, pages 313–317, 2011b.

Veli Mäkinen and Gonzalo Navarro. Succinct suffix arrays based on run-length encoding. *Nordic J. of Computing*, 12(1):40–66, March 2005.

Veli Mäkinen and Gonzalo Navarro. Dynamic entropy-compressed sequences and full-text indexes. *ACM Trans. Algorithms*, 4(3):32:1–32:38, July 2008.

Udi Manber. Finding similar files in a large file system. In *in Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, 1994.

Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proc. of the 28th VLDB*, pages 346–357, 2002.

Mauricio Marin, Veronica Gil-Costa, and Carlos Gomez-Pantoja. New caching techniques for web search engines. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 215–226, 2010.

E.P Markatos. On caching search engine query results. *Computer Communications*, 24(2):137 – 143, 2001.

K. Tamsin Maxwell and W. Bruce Croft. Compact query term selection using topically related text. In *Proc. of the 2011 ACM SIGIR*, pages 583–592, 2013.

D. Metzler and W.B. Croft. A markov random field model for term dependencies. In *Proc. of the 28th ACM SIGIR*, pages 472–479, 2005.

Donald Metzler. Using gradient descent to optimize language modeling smoothing parameters. In *Proc. of the 30th ACM SIGIR*, pages 687–688, 2007.

Donald Metzler, Trevor Strohman, and W. Bruce Croft. A statistical view of binned retrieval models. In *Proc. of the 30th ECIR*, pages 175–186, 2008.

Gilad Mishne and Maarten de Rijke. Boosting web retrieval through query operations. In *Proc. of the 27th ECIR*, pages 502–516, 2005.

Alistair Moffat and Justin Zobel. Index organization for multimedia database systems. *ACM Comput. Surv.*, 27(4):607–609, December 1995.

Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst.*, 14(4):349–379, October 1996.

Alistair Moffat and Justin Zobel. What does it mean to "measure performance"? In *Proc. 5th Int. Conf. on Web Informations Systems*, pages 1–12. LNCS 3306, Springer, November 2004.

S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *SODA*, pages 657–666, 2002.

S. Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers, 2005.

Ramesh Nallapati and James Allan. Capturing term dependencies using a language model based on sentence trees. In *Proceedings of the 11th CIKM*, pages 383–390, 2002.

G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):2–1 – 2–61, 2007.

G. Navarro and D. Valenzuela. Space-efficient top-k document retrieval. In *SEA*, LNCS 7276, pages 307–319, 2012.

I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A High Performance and Scalable Information Retrieval Platform. In *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*, 2006.

Rifat Ozcan, I. Sengor Altingovde, B. Barla Cambazoglu, Flavio P. Junqueira, and ÖZgüR Ulusoy. A five-level static cache architecture for web search engines. *Inf. Process. Manage.*, 48(5):828–840, September 2012.

Jae Hyun Park, W. Bruce Croft, and David A. Smith. A quasi-synchronous dependence model for information retrieval. In *CIKM*, pages 17–26, 2011.

M. Patil, S. V. Thankachan, R. Shah, W.-K. Hon, J. S. Vitter, and S. Chandrasekaran. Inverted indexes for phrases and strings. In *SIGIR*, pages 555–564, 2011.

Jie Peng, Craig Macdonald, Ben He, Vassilis Plachouras, and Iadh Ounis. Incorporating term dependency in the dfr framework. In *Proc. of the 30th ACM SIGIR*, pages 843–844, 2007.

Michael Persin, Justin Zobel, and Ron Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *J. Am. Soc. Inf. Sci.*, 47(10):749–764, September 1996.

Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proc. of the 21st ACM SIGIR*, pages 275–281, 1998.

S. J. Puglisi, W. F. Smyth, and A. H. Turpin. A taxonomy of suffix array construction algorithms. *ACM Computing Surveys*, 39(2):4.1–4.31, 2007.

Fiana Raiber and Oren Kurland. Ranking document clusters using markov random fields. In *SIGIR*, pages 333–342, 2013.

Yves Rasolofo and Jacques Savoy. Term proximity scoring for keyword-based retrieval systems. In *Proceedings of the 25th ECIR*, pages 207–218, 2003.

Knut Magne Risvik, Tomasz Mikolajewski, and Peter Boros. Query segmentation for web search. In *Proc. of the 12th WWW*, 2003.

S. E. Robertson and K. Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proc. of the 17th ACM SIGIR*, pages 232–241, 1994.

Stephen E. Robertson, Steve Walker, Micheline H. Beaulieu, Aarron Gull, and Marianna Lau. Okapi at trec-3. In *TREC-3*, pages 21–30, 1992.

Ron Sacks-Davis, Alan J. Kent, Kotagiri Ramamohanarao, James A. Thom, and Justin Zobel. Atlas: A nested relational database system for text applications. *IEEE Trans. Knowl. Data Eng.*, 7(3):454–470, 1995.

K. Sadakane. Succinct data structures for flexible text retrieval systems. *J. Discr. Alg.*, 5(1):12–22, 2007.

G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, August 1988.

P. Sanders and F. Transier. Intersection in integer inverted indices. In *Proc. 10th ALENEX Workshop on Algorithm Engineering and Experiments*, pages 71–83. Siam, January 2007.

Paricia Correia Saraiva, Edleno Silva de Moura, Novio Ziviani, Wagner Meira, Rodrigo Fonseca, and Berthier Riberio-Neto. Rank-preserving two-level caching for scalable search engines. In *Proceedings of the 24th SIGIR*, pages 51–58, 2001.

Ralf Schenkel, Andreas Broschart, Seungwon Hwang, Martin Theobald, and Gerhard Weikum. Efficient text proximity search. In *Proceedings of the 14th SPIRE*, pages 287–299, 2007.

Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proc. of the 2003 ACM SIGMOD*, pages 76–85, 2003.

J. Seo and W.B. Croft. Local text reuse detection. In *Proc. of the 31st ACM SIGIR*, pages 571–578. ACM, 2008.

Lixin Shi and Jian-Yun Nie. Using various term dependencies according to their utilities. In *Proc. of the 19th ACM CIKM*, pages 1493–1496, 2010.

Mark D. Smucker, James Allan, and Ben Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proc. of the 16th ACM CIKM*, pages 623–632, 2007.

Fei Song and W. Bruce Croft. A general language model for information retrieval. In *Proc. of the eighth CIKM*, pages 316–321, 1999.

Ruihua Song, Michael J. Taylor, Ji-Rong Wen, Hsiao-Wuen Hon, and Yong Yu. Viewing term proximity from a different perspective. In *Proc. 30th ECIR*, pages 346–357, 2008.

Munirathnam Srikanth and Rohini Srihari. Incorporating query term dependencies in language models for document retrieval. In *Proc. of the 26th ACM SIGIR*, pages 405–406, 2003.

Trevor Strohman and W. Bruce Croft. Efficient document retrieval in main memory. In *Proc. of the 30th ACM SIGIR*, pages 175–182, 2007.

Trevor Strohman, Howard Turtle, and W. Bruce Croft. Optimization strategies for complex queries. In *Proceedings of the 28th ACM SIGIR*, pages 219–225, 2005.

Krysta M. Svore, Pallika H. Kanani, and Nazan Khan. How good is a span of terms?: exploiting proximity to improve web retrieval. In *Proceedings of the 33rd ACM SIGIR*, pages 154–161, 2010.

Tao Tao and ChengXiang Zhai. An exploration of proximity measures in information retrieval. In *Proc. of the 30th ACM SIGIR*, pages 295–302, 2007.

N. Thaper, P. Indyk, S. Guha, and N. Koudas. Dynamic multiminesional histograms. In *Proc. of the 2002 ACM SIGMOD*, pages 428–439, 2002.

The Lemur Project. http://www.lemurproject.org/. Lemur Toolkit, Indri, Galago, ClueWeb09, ClueWeb12, 2001-2013.

F. Transier and P. Sanders. Out of the box phrase indexing. In *SPIRE*, LNCS 5820, pages 200–211, 2008.

F. Transier and P. Sanders. Engineering basic algorithms of an in-memory text search engine. *ACM TOIS*, 29(1):2–1–2–37, 2010.

A. Tridgell and R. P. Brent. An implementation of a general-purpose parallel sorting algorithm. Technical Report TR-CS-93-01, Australian National University, 1993. URL `http://gan.anu.edu.au/~brent/pd/rpb140tr.pdf`.

Howard Turtle and James Flood. Query evaluation: Strategies and optimizations. *Information Processing and Management*, 31(6):831 – 850, 1995.

Howard R Turtle, Gerald J Morton, and F Kinley Larntz. System of document representation retrieval by successive iterated probability sampling, 01 1996. US Patent 5,488,725.

C. J. van Rijsbergen. A theoretical basis for the use of co-occurrence data in information retrieval. *Journal of Documentation*, 33(2):106–199, 1977.

Lidan Wang, Jimmy Lin, and Donald Metzler. Learning to efficiently rank. In *Proceedings of the 33rd ACM SIGIR*, pages 138–145, 2010.

W. Webber and A. Moffat. In search of reliable retrieval experiments. In *Proc. of the 10th ADCS*, pages 26–33, 2005.

Hugh E. Williams and Justin Zobel. Compressing integers for fast file access. *The Computer Journal*, 42:193–201, 1999.

Hugh E. Williams, Justin Zobel, and Dirk Bahle. Fast phrase querying with combined indexes. *ACM Trans. Inf. Syst.*, 22(4):573–594, October 2004.

I.H. Witten, A. Moffat, and T.C. Bell. *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1999.

S. K. M. Wong, Wojciech Ziarko, and Patrick C. N. Wong. Generalized vector spaces model in information retrieval. In *Proc. of the 8th ACM SIGIR*, SIGIR '85, pages 18–25, 1985.

X. Xue and W.B. Croft. Representing queries as distributions. In *SIGIR10 Workshop on Query Representation and Understanding*, pages 9–12, 2010.

Xiaobing Xue and W. Bruce Croft. Modeling subset distributions for verbose queries. In *Proc. of the 34th ACM SIGIR*, pages 1133–1134, 2011.

Xiaobing Xue and W. Bruce Croft. Generating reformulation trees for complex queries. In *Proceedings of the 35th ACM SIGIR*, pages 525–534, 2012.

Xiaobing Xue, Samuel Huston, and W. Bruce Croft. Improving verbose queries using subset distribution. In *Proc. of the 19th ACM CIKM*, pages 1059–1068, 2010.

Hao Yan, Shuming Shi, Fan Zhang, Torsten Suel, and Ji-Rong Wen. Efficient term proximity search with term-pair indexes. In *Proceedings of the 19th ACM CIKM*, CIKM '10, pages 1229–1238, 2010.

C.T. Yu, C. Buckley, K. Lam, and G. Salton. A generalized term dependence model in information retrieval. Technical report, Cornell University, 1983.

J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490, December 1998.

Marcin Zukowski, Sandor Heman, Niels Nes, and Peter Boncz. Super-scalar ram-cpu cache compression. In *Proc. of the 22nd ICDE*, pages 59–, 2006.